UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

OPTIMIZACIÓN DE LA ADQUISICIÓN DE MODELOS DE OPONENTES EN LA ROBOCUP – CATEGORÍA DE SIMULACIÓN 2D

Autor: Alberto Blanco Martínez

Tutor: Agapito Ledezma Espino

Mayo, 2012

AGRADECIMIENTOS

A mi tutor, por su paciencia y por su disponibilidad; a todos los miembros del grupo CAOS por su continuo ofrecimiento a ayudarme; a mi familia por su apoyo silencioso; a mis amigos y compañeros de universidad por su continuo interés; y a Marisa, por su ayuda en todos los ámbitos de la vida.

ÍNDICE

1 INTRODUCCIÓN 1

1.1 CONTEXTO 1
1.2 OBJETIVOS 2
1.3 ESTRUCTURA DEL DOCUMENTO 3

2 ESTADO DEL ARTE 4

2.1 ¿QUÉ ES UN AGENTE? 4
2.2 SISTEMAS MULTIAGENTE 6
2.3 MODELADO DE AGENTES 7
2.4 ROBOCUP 10
2.4.1 Categorías de la RoboCup Soccer 11
2.4.2 Otras iniciativas de la RoboCup 15
2.5 ROBOCUP SOCCER SERVER 15
2.5.1 Características del Soccer Server 16
2.5.2 Funcionamiento del Soccer Server 17
2.6 EQUIPO CMUNITED 99 20
2.6.1 Arquitectura del agente 20
2.6.2 Modelado del mundo 22
2.6.3 Habilidades del agente 23
2.7 MODELADO DE AGENTES BASADO EN LA OBSERVACIÓN 24

3 PLANTEAMIENTO Y DESARROLLO DEL PROYECTO 28

3.1 INTRODUCCIÓN 28
3.2 OPTIMIZACIÓN DEL PROCESO DE MODELADO 28
3.2.1 Recolección de datos basada en la observación 28
3.2.2 Generación del etiquetador 33
3.2.3 Generación del modelador 37
3.2.4 Configuración experimental 42

4 RESULTADOS EXPERIMENTALES 46

4.1 OPTIMIZACIÓN DEL PROCESO DE MODELADO 46
4.1.1 Verificaciones realizadas 46
4.1.2 Resultado final de la optimización del modelo 49
4.2 ADQUISICIÓN Y VERIFICACIÓN DE MODELOS 50
4.2.1 Clientes utilizados 50
4.2.2 Consideraciones iniciales 51
4.2.3 Modelado del agente UvA Trilearn 54
4.2.4 Modelado del agente YowAI 57
4.2.5 Modelado del agente Virtual werder 60
4.2.6 Modelado del agente Wright Eagle 62
4.2.7 Modelado del agente Tokio Tech 64
4.2.8 Modelado del agente FC Portugal 66
4.2.9 Comparativa de resultados 68
4.2.10 Conclusiones de la experimentación 70

5 CONCLUSIONES GENERALES 73

6 FUTURAS LÍNEAS DE DESARROLLO 74

7 BIBLIOGRAFÍA 75

ANEXO A PLANIFICACIÓN Y PRESUPUESTO 78

A.1 PLANIFICACIÓN 78
A.2 PRESUPUESTO 80
A.2.1 Desglose por actividades 80
A.2.2 Coste de personal 80
A.2.3 Coste de hardware 81
A.2.4 Coste de software y licencias 81
A.2.5 Coste de material fungible 82
A.2.6 Resumen de costes 82
A.2.7 Plantilla de presupuesto 83

ANEXO B MANUAL DE USUARIO 84

B.1 REQUISITOS DEL SISTEMA 84
B.2 CONTENIDO A INSTALAR 84
B.3 PROGRAMAS PROPIOS DEL PROYECTO 87
B.3.1 Agentes de la RoboCup 87
B.3.2 Programas para extraer información de los logs 87
B.3.3 Ficheros de configuración externos 88
B.4 CREACIÓN DEL ETIQUETADOR 91
B.4.1 Obtención de datos 92
B.4.2 Modificación del log generado por el servidor 96
B.4.3 Fusión de los datos de ambos agentes 98
B.4.4 Generación de los archivos propios de Weka 100
B.4.5 Procesamiento en Weka 102
B.4.6 Inclusión de los ficheros de reglas en el etiquetador 112
B.5 CREACIÓN DEL MODELADOR 117
B.5.1 Obtención de datos 118
B.5.2 Generación de un fichero con los datos sensados por el agente CMUnited 99 119
B.5.3 Generación de los archivos propios de Weka y posterior procesamiento 120
B.5.4 Inclusión de los ficheros de reglas en el etiquetador 120

ANEXO C MANUAL DE REFERENCIA 122

C.1 LISTADO DE FICHEROS 122
C.1.1 Ficheros del cliente CMUnited 99 122
C.1.2 Ficheros del Trainer 123
C.1.3 Otros ficheros 124
C.2 SCRIPTS UTILIZADOS 125
C.3 MODIFICACIONES EN LOS AGENTES DE LA ROBOCUP 129

ÍNDICE DE FIGURAS

FIGURA 1: EL AGENTE Y SU ENTORNO 5
FIGURA 2: ROBOTS DE CATEGORÍA HUMANOIDE 12
FIGURA 3: ROBOTS DE LA PLATAFORMA ESTÁNDAR 13
FIGURA 4: PARTIDO DE LA ROBOCUP DE LA CATEGORÍA MEDIA 13
FIGURA 5: PARTIDO DE LA ROBOCUP DE LA CATEGORÍA DE TAMAÑO PEQUEÑO 14
FIGURA 6: SIMULADOR DEL CAMPO 18
FIGURA 7: VISTA GENERAL DEL *SOCCER SERVER* 19
FIGURA 8: MODELO FUNCIONAL DE E/S DE LA ARQUITECTURA DEL AGENTE CMUNITED 99 21
FIGURA 9: CREACIÓN DEL MÓDULO DE MODELADO DE AGENTES BASADO EN LA OBSERVACIÓN 25
FIGURA 10: CREACIÓN DEL MÓDULO DE ETIQUETADO DE ACCIONES 26
FIGURA 11: VISIÓN DEL AGENTE 32
FIGURA 12: ESQUEMA DEL PROCESO DE CREACIÓN DEL ETIQUETADOR 33
FIGURA 13: IMPLEMENTACIÓN DEL MÓDULO DE ETIQUETADO DE ACCIONES 36
FIGURA 14: ESQUEMA DEL PROCESO DE CREACIÓN DEL MODELADOR 38
FIGURA 15: IMPLEMENTACIÓN DEL MÓDULO DE CONSTRUCCIÓN DEL MODELO 41
FIGURA 16: COLOCACIÓN ALEATORIA DEL JUGADOR Y DE LA PELOTA 43
FIGURA 17: VARIACIÓN EN LA SIMULACIÓN: GOLES ANOTADOS 68
FIGURA 18: VARIACIÓN EN LA SIMULACIÓN: TIROS FUERA 69
FIGURA 19: VARIACIÓN EN LA SIMULACIÓN: MEJORÍA AL UTILIZAR UN CLASIFICADOR MÁS PRECISO 72
FIGURA 20: PLANIFICACIÓN DEL PROYECTO: DIAGRAMA DE GANTT PARTE I 79
FIGURA 21: PLANIFICACIÓN DEL PROYECTO: DIAGRAMA DE GANTT PARTE II 79
FIGURA 22: PLANTILLA DE PRESUPUESTO 83
FIGURA 23: ESQUEMA DE IMPLEMENTACIÓN DEL ETIQUETADOR 91
FIGURA 24: EJECUCIÓN DEL AGENTE TRAINER 94
FIGURA 25: CAMPO DE JUEGO VISTO A TRAVÉS DEL ROBOCUP SOCCER MONITOR 95
FIGURA 26: WEKA EXPLORER: CARGA DEL FICHERO *PRINCIPAL.ARFF* 103
FIGURA 27: WEKA EXPLORER: CLASE DEL FICHERO DE TIPO PRINCIPAL 105
FIGURA 28: WEKA EXPLORER: OPCIONES DEL CLASIFICADOR PART 107
FIGURA 29: WEKA EXPLORER: RESULTADOS DEL CLASIFICADOR PART 107
FIGURA 30: WEKA EXPLORER: CLASE DEL FICHERO DE TIPO DASH/TURN 110
FIGURA 31: WEKA EXPLORER: CLASIFICADOR M5 111
FIGURA 32: ESQUEMA DE IMPLEMENTACIÓN DEL MODELADOR 117

ÍNDICE DE TABLAS

TABLA 1: ATRIBUTOS SENSADOS POR EL AGENTE 30
TABLA 2: ATRIBUTOS CALCULADOS POR EL AGENTE 31
TABLA 3: TIEMPO EMPLEADO EN LA SIMULACIÓN 53
TABLA 4: RESULTADOS DE LA CREACIÓN DEL ETIQUETADOR PARA EL AGENTE UVA TRILEARN 55
TABLA 5: RESULTADOS DE LA CREACIÓN DEL MODELADOR PARA EL AGENTE UVA TRILEARN 55
TABLA 6: RESULTADOS DE LA SIMULACIÓN SIN MODELO DEL AGENTE UVA TRILEARN 55
TABLA 7: RESULTADOS DE LA SIMULACIÓN CON MODELO DEL AGENTE UVA TRILEARN 56
TABLA 8: GANANCIA EN LA SIMULACIÓN DEBIDA A LA INCLUSIÓN DEL MODELO EN EL AGENTE UVA TRILEARN 56
TABLA 9: RESULTADOS DE LA CREACIÓN DEL ETIQUETADOR PARA EL AGENTE UVA TRILEARN ORIGINAL - MODIFICADO 56
TABLA 10: RESULTADOS DE LA CREACIÓN DEL MODELADOR PARA EL AGENTE UVA TRILEARN ORIGINAL - MODIFICADO 57
TABLA 11: RESULTADOS DE LA SIMULACIÓN SIN MODELO DEL AGENTE UVA TRILEARN ORIGINAL - MODIFICADO 57
TABLA 12: RESULTADOS DE LA SIMULACIÓN CON MODELO DEL AGENTE UVA TRILEARN ORIGINAL - MODIFICADO 57
TABLA 13: RESULTADOS DE LA CREACIÓN DEL ETIQUETADOR PARA EL AGENTE YOWAI 58
TABLA 14: RESULTADOS DE LA CREACIÓN DEL MODELADOR PARA EL AGENTE YOWAI 58
TABLA 15: RESULTADOS DE LA SIMULACIÓN SIN MODELO DEL AGENTE YOWAI 59
TABLA 16: RESULTADOS DE LA SIMULACIÓN CON MODELO DEL AGENTE YOWAI 59
TABLA 17: GANANCIA EN LA SIMULACIÓN DEBIDA A LA INCLUSIÓN DEL MODELO EN EL AGENTE YOWAI 59
TABLA 18: RESULTADOS DE LA CREACIÓN DEL ETIQUETADOR PARA EL AGENTE VIRTUAL WERDER 60
TABLA 19: RESULTADOS DE LA CREACIÓN DEL MODELADOR PARA EL AGENTE VIRTUAL WERDER 60
TABLA 20: RESULTADOS DE LA SIMULACIÓN SIN MODELO DEL AGENTE VIRTUAL WERDER 61
TABLA 21: RESULTADOS DE LA SIMULACIÓN CON MODELO DEL AGENTE VIRTUAL WERDER 61
TABLA 22: GANANCIA EN LA SIMULACIÓN DEBIDA A LA INCLUSIÓN DEL MODELO EN EL AGENTE VIRTUAL WERDER 61
TABLA 23: RESULTADOS DE LA CREACIÓN DEL ETIQUETADOR PARA EL AGENTE WRIGHT EAGLE 62
TABLA 24: RESULTADOS DE LA CREACIÓN DEL MODELADOR PARA EL AGENTE WRIGHT EAGLE 62
TABLA 25: RESULTADOS DE LA SIMULACIÓN SIN MODELO DEL AGENTE WRIGHT EAGLE 63
TABLA 26: RESULTADOS DE LA SIMULACIÓN CON MODELO DEL AGENTE WRIGHT EAGLE 63
TABLA 27: GANANCIA EN LA SIMULACIÓN DEBIDA A LA INCLUSIÓN DEL MODELO EN EL AGENTE WRIGHT EAGLE 63
TABLA 28: RESULTADOS DE LA CREACIÓN DEL ETIQUETADOR PARA EL AGENTE TOKIO TECH 64
TABLA 29: RESULTADOS DE LA CREACIÓN DEL MODELADOR PARA EL AGENTE TOKIO TECH 64
TABLA 30: RESULTADOS DE LA SIMULACIÓN SIN MODELO DEL AGENTE TOKIO TECH 65
TABLA 31: RESULTADOS DE LA SIMULACIÓN CON MODELO DEL AGENTE TOKIO TECH 65
TABLA 32: GANANCIA EN LA SIMULACIÓN DEBIDA A LA INCLUSIÓN DEL MODELO EN EL AGENTE TOKIO TECH 65
TABLA 33: RESULTADOS DE LA CREACIÓN DEL ETIQUETADOR PARA EL AGENTE FC PORTUGAL 66
TABLA 34: RESULTADOS DE LA CREACIÓN DEL MODELADOR PARA EL AGENTE FC PORTUGAL 66
TABLA 35: RESULTADOS DE LA SIMULACIÓN SIN MODELO DEL AGENTE FC PORTUGAL 67
TABLA 36: RESULTADOS DE LA SIMULACIÓN CON MODELO DEL AGENTE FC PORTUGAL 67
TABLA 37: GANANCIA EN LA SIMULACIÓN DEBIDA A LA INCLUSIÓN DEL MODELO EN EL AGENTE FC PORTUGAL 67
TABLA 38: COSTE DE PERSONAL 81
TABLA 39: COSTE DE HARDWARE 81
TABLA 40: COSTE DE SOFTWARE Y LICENCIAS 81
TABLA 41: COSTE DE MATERIAL FUNGIBLE 82
TABLA 42: RESUMEN DE COSTES 82
TABLA 43: REPRESENTACIÓN NUMÉRICA DE LAS DIFERENTES ACCIONES POSIBLES 100

1 INTRODUCCIÓN

1.1 CONTEXTO

Una de las ramas más importantes de la informática es la Inteligencia Artificial (IA), y un subconjunto cada vez más importante lo forma la Inteligencia Artificial Distribuida (IAD), donde se trabaja con los sistemas de agentes inteligentes para resolver problemas complejos, donde un agente puede ser un agente software, o un agente hardware o robot.

La teoría del aprendizaje social [1], que proporciona la base para el modelado del comportamiento, asegura que la mayoría de las conductas a nivel social se aprenden mediante la observación. Esta teoría se aplica tanto a las personas como a los agentes, y desemboca en proporcionar la capacidad de predecir las acciones futuras que se darán en el entorno en el que se encuentra, lo cual es de vital importancia para conseguir la mejor adaptación posible.

En este proyecto se tratará el tema del modelado de agentes, basándose en la extracción y representación del conocimiento sobre el comportamiento de un agente en un entorno competitivo. En este tipo de dominios en los que otros agentes interfieren en el cumplimiento de las metas, Riley y Veloso [2] aseguran que los agentes han de ser capaces de adaptarse al comportamiento de los oponentes si pretenden tener éxito.

Este proyecto se centra en el entorno futbolístico, partiendo de las bases definidas por la competición *RoboCup*, una iniciativa de investigación y educación a nivel internacional que provee una competición de fútbol que se realiza anualmente, y que cuenta con un amplio respaldo por parte de múltiples universidades y asociaciones.

El objetivo final de este proyecto es determinar la influencia que puede tener el uso del modelo de un oponente a la hora de interactuar con otro agente. Parte de una idea existente, que es el trabajo realizado en la tesis doctoral “Aprendizaje automático en conjuntos de clasificadores heterogéneos y modelado de agentes” [3], y continúa el trabajo en el punto donde terminó. La forma de determinar dicha influencia ha sido a través de la evolución de un agente de tipo delantero adaptándose al comportamiento de un agente de tipo portero a través de la información obtenida a través de la observación.

Este proyecto propone la utilización de una serie de herramientas creadas específicamente para optimizar dicho proceso, y gracias a las utilidades que proporciona la *RoboCup* este tipo de procedimientos son fácilmente verificables, lo cual ayudará a la hora de determinar si el agente al que se ha añadido el modelo del oponente se ha adaptado mejor o no al entorno.

1.2 OBJETIVOS

El objetivo de este proyecto es el de avanzar en el conocimiento del modelado de agentes basado en la observación. Este proceso se hace a través de la optimización de un proceso existente enfocado en la creación de un modelo de comportamiento en sistemas multiagente, y de la posterior experimentación para verificar dicho proceso.

A partir de esta premisa se puede declarar que el objetivo del proyecto es doble, y como tal la memoria será estructurada en consecuencia.

1. El primer objetivo es implementar el modelo de aprendizaje basado en la observación (MABO) a través de las herramientas que se han generado, mostrando el camino del modelo teórico a la implementación práctica.
2. El segundo objetivo es verificar los resultados obtenidos por dicho modelo compitiendo contra agentes reales, demostrando su validez de forma cuantificable.

Para llevar a cabo estos objetivos, se ha definido una serie de tareas a realizar durante el desarrollo del proyecto, que son las siguientes:

- Comprensión de la estructura del entorno RoboCup, elementos de los que consta y de los agentes que participan en dicho entorno, en concreto de un agente que logró la victoria en el campeonato en el que participó.
- Dotación de un método de obtención de datos a dicho agente basado en la observación del comportamiento del agente rival.
- Tratamiento de dichos datos de forma que sea posible generar un modelo de comportamiento del agente rival.
- Creación de una serie de utilidades intermedias para dicho tratamiento, entre las que se incluyen herramientas para convertir los datos observados en ficheros de datos comprensibles por herramientas de análisis.
- Disposición de métodos de medición de resultados, tarea necesaria para calibrar el grado de mejora que supone la incorporación del modelo de comportamiento del agente rival al agente delantero, únicamente verificable a través de la experimentación.
- Establecimiento de una metodología a seguir para la posterior utilización de los procedimientos establecidos en este proyecto para futuros trabajos.

1.3 ESTRUCTURA DEL DOCUMENTO

Este proyecto está estructurado del siguiente modo:

- En el capítulo dos se hace una introducción a los agentes en general y al modelado de agentes en particular. Además se explica la iniciativa RoboCup y sus diferentes categorías, entre ellas la utilizada en este proyecto, la categoría de simulación. Por último, trata el sistema de modelado de agentes basado en la observación, base de este trabajo y que se tendrá en cuenta en cada apartado.
- En el capítulo tres se muestra el planteamiento y el desarrollo del trabajo realizado. Este capítulo hace referencia a uno de los objetivos básicos de este proyecto, concretamente la optimización del proceso de implementación del MABO.
- El capítulo cuatro hace referencia al otro punto objetivo central de este proyecto, que es la experimentación y verificación de resultados, haciendo competir el agente desarrollado contra otros agentes antes y después de haber implementado el modelo de comportamiento del agente rival generado.
- En el capítulo cinco se desarrollan las conclusiones a las que se ha llegado, tanto de la experimentación como del proyecto en general.
- El capítulo seis referencia las futuras líneas de desarrollo de este proyecto.
- En el Anexo A se encuentra la planificación del proyecto, que permite seguir de un modo gráfico el tiempo invertido en cada una de las tareas de las que consta este proyecto. Además contiene el presupuesto de los costes asociados al proyecto, con una descripción de las distintas tareas realizadas y el tiempo invertido en cada una de ellas.
- En el Anexo B se encuentra el manual de usuario. Debido a la complejidad del proceso de implementación del MABO, este manual es indispensable para poder hacer uso de todas las utilidades que se han desarrollado.
- En el Anexo C se dispone del manual de referencia. Contiene información específica de la implementación. Este manual está orientado a personas que deseen continuar el proyecto.

2 ESTADO DEL ARTE

El objetivo de este capítulo es el de proporcionar una panorámica del estado actual del modelado de agentes y de la iniciativa RoboCup, facilitando un contexto al proyecto desarrollado.

2.1 ¿QUÉ ES UN AGENTE?

Para empezar se ha de determinar qué es un agente. Etimológicamente hablando la palabra proviene del latín *agere*, que significa "el que hace". Difícilmente se puede encontrar una explicación que sea globalmente aceptada por toda la comunidad que trabaja en agentes, puesto que se pretende definir un término utilizado en multitud de áreas diferentes.

Woolridge y Jennings [4] asumen que se puede contar con dos nociones de agentes, una más débil en la que un agente se puede definir de acuerdo a las características de *autonomía*, *habilidad social*, *reactividad* o capacidad de reaccionar y *proactividad* o capacidad de tomar la iniciativa; y una definición más fuerte en el sentido de que tiene un significado más específico, explicación que proviene esencialmente de los investigadores en el área de Inteligencia Artificial (IA), los cuales observan a los agentes con características antropomórficas, usualmente aplicadas a los humanos, como pueden ser la *movilidad*, la *veracidad*, la *benevolencia* y la *racionalidad*.

En términos generales, se puede decir que un agente es una entidad inteligente que existe dentro de un cierto contexto o ambiente, con el cual puede interactuar, y además se puede comunicar con otros agentes en su mismo entorno.

Macal y North [5], consideran que un agente tiene que tener las siguientes características:

- Un agente ha de ser un individuo identificable, con un conjunto de características y reglas que definen sus comportamientos, y la capacidad de tomar decisiones.
- Un agente ha de ser autocontenido, de tal forma que tenga unos límites que permitan diferenciar lo que es parte de un agente, lo que no, y lo que está compartido por otros agentes.
- Un agente ha de ser autónomo, puede funcionar de forma independiente en su entorno y en sus interacciones con otros agentes.
- Un agente ha de ser social, con la habilidad de interactuar con otros agentes, a través de protocolos de comunicación establecidos.
- Un agente debe estar situado, es decir, debe “vivir” en un entorno externo a él con el que interactúa, junto con otros agentes.
- Un agente debe estar orientado a una meta, tiene una serie de objetivos hacia los que van encaminados sus comportamientos.

- Un agente ha de ser flexible, teniendo la habilidad de aprender y adaptar sus comportamientos en base a la experiencia. Esta característica requiere de algún tipo de memoria interna.

La Figura 1 muestra un esquema de cómo debería en principio ser un agente, entendido como una entidad autónoma con sus características de interacción tanto con el entorno como con otros agentes, y con toda la información y reglas internas que maneja:

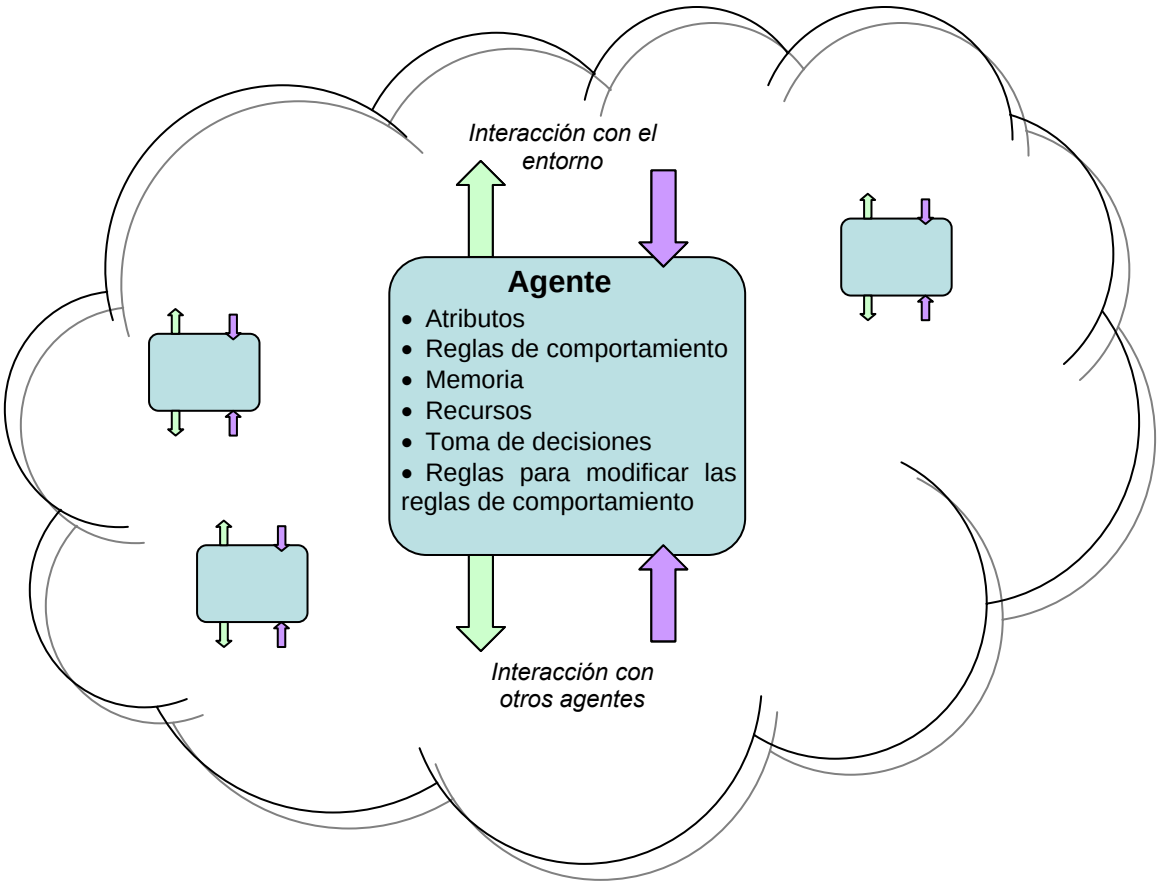


Figura 1: El agente y su entorno

Existen multitud de campos de utilización de agentes en la actualidad. Por poner unos ejemplos, los principales buscadores de Internet construyen sus bases de datos usando agentes del tipo *spiders* o *crawlers*, que recorren diferentes páginas web de forma automática para más tarde indexarlas, como pueden ser GoogleBot [19], o Yahoo! Slurp [20].

También se utilizan agentes para la predicción metereológica, e incluso para realizar simulaciones y previsiones del comportamiento oceánico, como es el caso del proyecto Coriolis [21].

Hay otros proyectos de agentes destinados al uso cotidiano, como el aspirador inteligente Roomba [22], que fue puesto a la venta en 2002 y ya va por su séptima revisión.

2.2 SISTEMAS MULTIAGENTE

La mayoría de los entornos en los que se trabaja con agentes precisan de la interacción y colaboración entre múltiples agentes para resolver problemas cada vez más complejos. Éste es el principio de la Inteligencia Artificial Distribuida (IAD), en la que se encuentran dos principales áreas de desarrollo, la solución cooperativa de problemas distribuidos (SCPD), o los sistemas multiagente (SMA).

Un sistema SCPD consta de un conjunto de módulos con tareas predefinidas que a través de la cooperación consiguen dividir un problema para llegar a una solución eficientemente. Este tipo de sistemas no utiliza agentes como tal puesto que son módulos dependientes entre sí, es decir, aunque interaccionan con el entorno carecen de una autonomía propia.

Un SMA consta de un grupo de agentes inteligentes con cierta autonomía que interactúan entre sí para llegar a la consecución de una serie de objetivos. La diferencia con SCPD es que estos objetivos pueden ser de tipo individual o colectivo, es decir, no tienen por qué tener objetivos comunes. Sycara [6] lo define como una red de agentes software que interactúan para resolver problemas que están más allá de las capacidades individuales o del conocimiento de cada agente individual.

Para Sycara [6], los principales problemas que se encuentran en este ámbito son los siguientes:

- Descomposición adecuada de los problemas para asignar tareas específicas a agentes individuales.
- Los agentes tienen una visión limitada del entorno, las decisiones a tomar se basan en una lógica del mundo incompleta.
- No existencia de una coordinación global de la actuación de los agentes y de las comunicaciones.
- Los datos a disposición de los agentes se encuentran descentralizados, es decir, no existe una organización de la que se pueda partir.
- Tratamiento de agentes con objetivos conflictivos.

Así pues, al tener cada agente una visión particular de su entorno, individualmente debe decidir cuál es la actuación adecuada a sus necesidades, y a la vez tratar de coordinarse con el resto de los agentes del sistema. Este comportamiento determina que una de las características más importantes de un SMA sea la modularidad, ya que el sistema visto desde una perspectiva exterior se compone de múltiples partes que interactúan entre sí para lograr su objetivo.

2.3 MODELADO DE AGENTES

Por modelado de agentes se entiende la habilidad individual de un agente de razonar sobre otros agentes con los que interactúa, permitiendo al agente discurrir sobre el comportamiento de un agente o grupo de agentes como conjunto de tal forma que se puedan predecir sus comportamientos futuros, con el objetivo de actuar en consecuencia.

En el campo de los sistemas multiagente, que se basa principalmente en la interacción entre agentes, es vital el conocimiento lo más preciso posible del comportamiento del resto de los agentes del entorno. Así pues, si el agente desea ser eficiente en su tarea, necesita obtener este tipo de información.

En el caso de modelado de oponentes, éste es el proceso en el que un agente genera un modelo de comportamiento de otro agente contra el que compite, a través de los medios de los que dispone. Habitualmente el agente únicamente contará con los sensores de que dispone para obtener la información, almacenando y tratando posteriormente dichos datos obtenidos en su memoria para poder predecir las acciones que tomará el agente rival según el estado actual del entorno. Así pues este tipo de modelado se basa principalmente en la observación. Si se dispone de conocimiento previo del agente a modelar, esta tarea resulta más sencilla, pero no es habitual este conocimiento.

El principal problema, como observan Kaminka et al. [7], es la ausencia de una certeza absoluta sobre el estado real del agente rival que se desea modelar, por lo que se hace necesario disponer del máximo conocimiento posible de dicho agente de forma previa, especialmente de las diferentes metas que persigue, para facilitar en la medida de lo posible la creación de un modelo que determine el comportamiento del rival.

Existe un área de la matemática aplicada denominada teoría de juegos, que se encarga de analizar los comportamientos estratégicos de los jugadores en los juegos, tomando como referencia una estructura formalizada de incentivos. Esta teoría tiene aplicaciones no sólo en el campo de los agentes software, sino también en ciencias económicas, ciencias políticas, estrategia militar, etc. La teoría de juegos estudia la elección de la conducta óptima cuando los costes y los beneficios de cada opción posible no están predefinidos, dependiendo de las elecciones de otros agentes que también participan en el juego. En cada turno cada jugador ha de seleccionar una acción de entre un conjunto determinado y finito de acciones posibles, así pues el jugador parte de la base de conocimiento de que conoce tanto las acciones que él mismo puede llevar a cabo como aquellas que pueden realizar el resto de jugadores.

La teoría de juegos nació en el año 1944 con el libro "*Theory of Games and Economic Behaviour*" escrito por John Von Neumann (que años después adquirió notoriedad por la definición de la arquitectura de las computadoras modernas) y Oskar Morgestern. Un ejemplo de aplicación de la teoría de juegos es el dilema del prisionero, en el que a dos presos se les ofrece colaborar

confesando o negando el crimen, a cambio de reducir la condena. En este caso concreto el resultado del juego depende de la elección de ambos jugadores, puesto que la coordinación entre ellos da el resultado óptimo: si ambos confiesan entonces la pena es la menor posible, si sólo uno de los dos confiesa entonces éste es condenado a la mayor pena y el otro es liberado, y si los dos niegan el crimen entonces la condena es de larga duración, pero menor que si uno delata al otro.

Este y otro tipo de juegos llevaron a la conclusión de que el conocimiento de la estrategia del rival es de gran importancia a la hora de elaborar la estrategia del propio jugador, por lo que se buscó la forma de crear modelos de comportamiento del oponente. Múltiples algoritmos fueron creados buscando este objetivo, como el método *Minimax* o la utilización de árboles de juego. Estos algoritmos precisan de un entorno muy concreto, con lo que no pueden ser utilizados en dominios complejos no deterministas en los que los individuos no juegan a través de turnos estrictos, como son la mayoría de los sistemas multiagentes.

Este proyecto se centra en el dominio de prueba de la competición de la RoboCup, que será explicada en profundidad en el punto 2.4. Este entorno proporciona un sistema multiagente de gran complejidad, por lo que los algoritmos básicos propuestos en la teoría de juegos no son eficientes.

Stone et al. [8] propone la utilización de clases predefinidas para la clasificación de los adversarios, estableciendo una serie de comportamientos óptimos a seguir por el agente basándose en un modelo del mundo ideal. Este modelo se denomina IMBBOP, *Ideal Model Based Behaviour Outcome Prediction*, y lo que hace es clasificar de forma estadística el comportamiento esperado del agente en términos de desviación respecto al comportamiento óptimo.

Riley y Veloso [2] definen una serie de puntos de obligado cumplimiento a la hora de obtener una clasificación del comportamiento de los adversarios, que son los siguientes:

1. Extracción de las características necesarias del estado del mundo en el que se encuentran los agentes. Este proceso es dependiente del entorno del sistema multiagente.
2. Construcción de una serie de clases de adversario a partir de la información contenida en las características extraídas en el punto 1. Estas clases deben contener la información estratégica sobre el comportamiento de cada tipo de adversario.
3. Comparación de las observaciones del adversario actual con las clases de adversario predefinidas en el punto 2, consiguiendo así una clasificación de los comportamientos del adversario.
4. Realización de cambios en el comportamiento de los agentes basados en la clasificación de los adversarios realizada en el punto 3, considerando las estrategias a desarrollar por los agentes dependiendo de las clases de adversarios establecidas.

Otro enfoque es el presentado por Steffens [9], que presenta el modelo FBDOM, *Feature-Based Declarative Opponent-Modelling*, en el que se procura identificar los movimientos tácticos del oponente. En este modelo el comportamiento de los adversarios se compara con modelos previamente almacenados de los oponentes, de tal forma que se clasifica el comportamiento basándose en análisis previos.

Otra orientación es la del modelo de reconocimiento de actividades, o *Activity recognition*, que tiene como objetivo reconocer las acciones y las metas de uno o más agentes a partir de unas series de observaciones de las acciones de los agentes, y de las condiciones del entorno. Este modelo tiene un primer acercamiento a través del reconocimiento de actividades a través de la lógica y el razonamiento, en los que se hace un seguimiento de todas las explicaciones lógicamente consistentes de las acciones observadas. De esta forma, todos los planes y metas posibles y consistentes deben ser considerados. Kautz [10] provee la primera teoría formal de reconocimiento de planes en la cual los planes y los objetivos se representan en una jerarquía de eventos que describe todo el comportamiento que puede exhibir un usuario en un dominio particular. Toda tarea observada es parte de uno o más planes de alto nivel, y el proceso de reconocimiento de planes consiste en minimizar el conjunto de planes de alto nivel suficientes para explicar las tareas observadas. El principal problema de este acercamiento es su incapacidad de representar la incertidumbre, debido a que no proporciona un mecanismo para priorizar entre varios planes consistentes, con lo que es incapaz de decidir qué plan es más probable que los demás, en el supuesto de que varios planes sean lo suficientemente consistentes para explicar las acciones observadas. Además este tipo de planteamiento carece de capacidad de aprendizaje, debido a que desde el primer momento se modela todo el comportamiento del agente a estudiar. Para paliar este tipo de defectos es posible la utilización de enfoques probabilísticos, como los modelos ocultos de Markov o las redes de Bayes.

La estructura de los modelos de Markov [11] es la de un grafo de transición de estados, en el que se presentan secuencias de eventos. El que se produzca la ocurrencia del siguiente evento depende sólo de una cantidad fija de eventos previos. Es decir, dado ciertos eventos previamente observados, el siguiente evento se predice a partir de la distribución de probabilidad de los eventos que siguieron a dichos eventos en el pasado.

Por su parte, las redes de Bayes [12] combinan una representación gráfica de relaciones causales con una base probabilística. Presentan una estructura de grafo dirigido acíclico que contiene representaciones tanto de las dependencias como de las independencias condicionales entre los elementos del dominio del problema. El conocimiento se representa por medio de nodos, también llamados variables, y de arcos, que representan las relaciones causales entre las variables. Además, se dispone de unas tablas de probabilidad condicional en las que se indica la probabilidad de que una determinada variable se encuentre en un determinado estado, partiendo de la configuración de los estados de las variables padre de esa determinada variable.

Por otra parte, Riley y Veloso [13] añaden un nuevo elemento para el modelado de comportamiento, el agente entrenador o *Coach*. Este agente especial es capaz de obtener información directa acerca de los agentes de un sistema multiagente, y basándose en las posiciones de los oponentes asociar el comportamiento de los rivales en su conjunto a una serie de modelos previamente definidos, pudiendo comunicarse con el resto de agentes de su mismo equipo para comunicarles los mejores planes posibles. Este enfoque se denomina ATAC, *Adaptative Team-Adversarial Coaching*.

Existe un enfoque adicional que propone la utilización de técnicas de aprendizaje automático para el modelado de agentes en el dominio de la RoboCup, estas técnicas son la base del presente proyecto, y son explicadas en profundidad en el punto 2.7.

2.4 ROBOCUP

La RoboCup [26] es una iniciativa internacional iniciada en 1995 para promover la Inteligencia Artificial y la robótica en un entorno común, en este caso un partido de fútbol. Es un intento de fomentar la inteligencia artificial y el desarrollo de la inteligencia robótica a través de un problema común donde un amplio rango de tecnologías pueden ser integradas y examinadas. La RoboCup eligió escoger el juego de fútbol como un tema central de desarrollo, con el objetivo de promover la investigación y la educación sobre inteligencia artificial.

El objetivo final del proyecto RoboCup es **"para el año 2050, desarrollar un equipo de robots humanoides completamente autónomos que pueda ganar al equipo humano campeón del mundo"**. Dicho objetivo es lo suficientemente ambicioso como para motivar año tras año la participación de diferentes equipos de todo el mundo. A día de hoy se muestra lejano tanto a nivel de hardware como de software, pero cada año se vienen realizando avances significativos.

Para que un agente tanto físico como software pueda jugar al fútbol, varias tecnologías deben ser incorporadas, entre las que se encuentran principios de diseño de agentes autónomos, colaboración en sistemas multiagente, adquisición de estrategias, razonamiento en tiempo real, todo ello con la base de los últimos avances a nivel de hardware.

El proyecto RoboCup ha tenido mucha repercusión a lo largo de su existencia, con decenas de participantes y amplia repercusión en los medios de comunicación. Los orígenes de esta iniciativa se remontan a 1995, durante la *International Joint Conference on Artificial Intelligence*, donde se realizó el anuncio de organizar la primera copa mundial de fútbol entre robots. En ese momento se tomó la decisión de organizar un evento previo, con el objetivo de identificar potenciales problemas asociados con la organización de la RoboCup a gran escala, además de proporcionar tiempo suficiente para el desarrollo de los programas y los robots participantes.

En noviembre de 2006 en Osaka, Japón, se celebró la Pre-RoboCup, en la que ocho equipos compitieron en una liga de simulación. Además se realizaron diversas demostraciones con robots

reales, en la que fue la primera competición que utilizó el juego de fútbol para promocionar la investigación y la educación.

La competición RoboCup se celebra anualmente desde 1997, siendo la primera sede la ciudad de Nagoya, Japón. En esta primera edición participaron más de 40 equipos entre las categorías de simulación y aquellas con robots reales. En el año 2011 se disputó en Estambul [27], Turquía, durante el mes de Julio. Dicha competición contó con más de 130 equipos en la sección de fútbol. Eventos anteriores fueron en Singapur, en el año 2010; Graz, Austria, en el año 2009; Suzhou, China en el año 2008; y en Atlanta, Estados Unidos, en el año 2007. La próxima competición se disputará en Ciudad de México en Junio de 2012. Lamentablemente esta competición todavía no se ha celebrado en España.

2.4.1 CATEGORÍAS DE LA ROBOCUP SOCCER

Existen varias categorías diferentes en la RoboCup. Con el paso de los años se ha separado la sección de *Soccer* o fútbol de otras iniciativas orientadas a otros entornos ajenos al fútbol (apartado 2.4.2). A continuación se definen las diferentes categorías de la competición RoboCup *Soccer*.

2.4.1.1 HUMANOIDE

La categoría *humanoid* o humanoide [28] es la más completa, consta a su vez de tres subcategorías: *KidSize* o tamaño de niño; *TeenSize* o tamaño de adolescente; y *AdultSize* o tamaño de adulto. Los robots de tipo *KidSize* han de medir entre 30 y 60cm de altura, los de tipo *TeenSize* varían entre los 100 y los 120cm de altura, y los *AdultSize* se sitúan en más de 130 cm. Esta categoría consta con multitud de restricciones en cuanto a la construcción de los robots, definiendo el tamaño máximo de pies, longitud de brazos y piernas, tamaño máximo alcanzable al expandirse, etc. El rectángulo de juego tiene unas dimensiones de 6 metros por 4 metros. Los partidos son, de como máximo, tres jugadores en cada equipo.

En la Figura 2 se muestra una composición de imágenes de las distintas categorías de robots de la liga humanoide de la competición realizada el año 2011. De izquierda a derecha se ven robots de la categoría *AdultSize*, de la categoría *TeenSize,* y de la categoría *KidSize*.

Figura 2: Robots de categoría Humanoide[1]

2.4.1.2 ESTÁNDAR

La categoría *standard platform* o estándar [30] se da únicamente con los robots humanoides *Nao* [31] fabricados por *Aldebaran Robotics*. Este tipo de máquinas sustituye a otras muy conocidas como son los robots con apariencia de perro llamados *Aibo* de la marca *Sony* [32], con los que compartieron competición hasta el año 2008, en el que se impuso el nuevo modelo. El robot *Nao* mide 57cm de altura y pesa alrededor de 4'5Kg, y su principal diferencia con el *Aibo* es que es de tipo bípedo. Ninguna modificación hardware es permitida en estos robots, únicamente se permiten personalizaciones en forma de logotipos de colores para distinguir los diferentes equipos.

El terreno de juego tiene dimensiones de 6'05 metros de longitud por 4'05 metros de anchura. Los partidos son jugados con tres robots en cada equipo.

En la Figura 3 se observa un ejemplo de partido de la competición llevada a cabo en el año 2011, en el que dos robots *Nao* están enfrentados. Los equipos se diferencian por unos elementos distintivos, en este caso de colores rojo y azul.

[1] Robots de categoría humanoide, fotografía extraída de la página web oficial de la RoboCup 2011 [27]

Figura 3: Robots de la plataforma estándar[2]

2.4.1.3 TAMAÑO MEDIO

La categoría *middle size* o tamaño medio [33] engloba robots que por volumen quepan en una caja de 50 cm de ancho por 50 cm de largo por 80 cm de altura, y tienen un peso máximo de 40 Kg. Las dimensiones del campo de juego son de 12 metros de longitud por 8 metros de anchura. Los partidos se dan en equipos de cinco jugadores incluyendo el portero.

La Figura 4 muestra un partido realizado en la competición del año 2011 de esta categoría.

Figura 4: Partido de la RoboCup de la categoría media[3]

[2] Robots de categoría estándar, fotografía extraída de la página web oficial de la RoboCup 2011 [27]

2.4.1.4 TAMAÑO PEQUEÑO

La categoría *small size* o tamaño pequeño [34] engloba los robots de menor tamaño. Cada robot debe ocupar en volumen como máximo un cilindro de 18 cm de diámetro y de 15 cm de altura. El campo de juego es de 6'05 metros de longitud por 4'05 metros de anchura. Los partidos son de cómo máximo cinco contra cinco agentes, incluyendo el portero. Esta categoría se centra principalmente en la cooperación entre los agentes en un entorno distribuido.

La Figura 5 permite observar un partido llevado a cabo en la competición del año 2011 de esta categoría.

Figura 5: Partido de la RoboCup de la categoría de tamaño pequeño[4]

2.4.1.5 SIMULACIÓN

Esta categoría de la RoboCup *Soccer* es la categoría relevante a este proyecto. Trata de simular el comportamiento de los agentes en un entorno también simulado. Consta de dos subcategorías diferentes, simulación en 2D (dos dimensiones) y en 3D (tres dimensiones), siendo más compleja esta última.

La categoría de simulación es óptima para implementar comportamientos de los agentes debido a que no es necesario realizar una inversión previa en hardware, y además una gran parte de los equipos libera el código fuente de sus agentes para el uso de la comunidad. Estas características hacen que la categoría de simulación esté muy extendida porque requiere un esfuerzo menor para participar en ella, centrándose principalmente en la inteligencia artificial y la estrategia de equipo.

[3] Robots de categoría media, fotografía extraída de la página web oficial de la RoboCup 2011 [27]
[4] Robots de categoría pequeña, fotografía extraída de la página web oficial de la RoboCup 2011 [27]

Este proyecto se engloba dentro de la categoría de simulación de la RoboCup, puesto que se trabaja únicamente con modelos software. En el punto 2.5 se explica ampliamente la infraestructura requerida para la categoría de simulación (también llamada *Soccer Server*), en concreto para la liga en 2D que es la utilizada en este proyecto.

2.4.2 OTRAS INICIATIVAS DE LA ROBOCUP

Además de la competición futbolística, otras iniciativas han surgido y actualmente forman parte de los eventos que se realizan. Los nuevos dominios son los siguientes:

- *RoboCupRescue*, liga creada en 2001 para promocionar el desarrollo de tecnologías de búsqueda y rescate a utilizar en desastres naturales de gran escala. Su objetivo principal es promover la investigación en aspectos socialmente significativos [35], tratando de crear un robot lo suficientemente inteligente y adaptable como para ser de utilidad en situaciones de catástrofes naturales. Esta competición consta de dos categorías: la liga de simulación y la liga con agentes físicos.
- *RoboCup@Home*, liga creada en 2008 y centrada en aplicaciones para el mundo real y basada en la interacción de los robots con los humanos. Su intencionalidad radica en el desarrollo de agentes que puedan servir en las tareas cotidianas.
- *RoboCupJunior*, iniciativa orientada a la enseñanza. Nace en el año 2000 con la intención de llamar la atención a los jóvenes de hasta 19 años. Esta iniciativa tiene diferentes competiciones, que son *Soccer*, *Rescue*, *Dance* (que busca el movimiento coordinado de pequeños robots al ritmo de la música), y desde el año 2009 una nueva competición denominada *CoSpace Demo Challenge* (orientada a la simulación, con una subliga orientada a la resolución de desafíos concretos llamada *CoSpace Adventure Challenge*, y otra dedicada al baile llamada *CoSpace Dance Challenge*).

2.5 ROBOCUP SOCCER SERVER

A la hora de trabajar con sistemas multiagente, surge la necesidad de disponer de una serie de recursos para simular el entorno, del tal forma que los agentes puedan interactuar con otros agentes o con el mismo entorno. Así pues, es necesaria una infraestructura base, tanto hardware como software, que posibilite el desarrollo de los agentes.

La infraestructura utilizada en la Liga de Simulación 2D de la RoboCup se llama *Soccer Server*. El *Soccer Server* tiene que aportar un entorno que simule de la forma más fidedigna posible un campo de fútbol real, por lo que tiene que poder soportar al menos a veintidós agentes autónomos actuando al mismo tiempo, simulando un campo de fútbol con sus dimensiones, sus líneas delimitadoras y una gran parte de sus normas. El *Soccer Server* plasma muchas de las complejidades del mundo real, como pueden ser condiciones limitadas a la hora de percibir el

entorno, acciones y movimientos de objetos con imprecisiones impuestas por el ruido externo, energía del agente limitada, así como una comunicación controlada y limitada entre agentes.

2.5.1 CARACTERÍSTICAS DEL SOCCER SERVER

Las características deseables para cualquier infraestructura utilizada en sistemas multiagentes, como comenta Gasser [14] en sus estudios en sistemas multiagentes, y con las que el *Soccer Server* cuenta, son las siguientes:

- Que esté disponible de forma gratuita. En el caso de la RoboCup, todos sus elementos están sujetos a la licencia LGPL, *GNU Lesser General Public License* [36], convirtiéndolo en software de dominio público con la peculiaridad de que puede ser incluido en aplicaciones no libres.
- Que cuente con un amplio soporte, tanto de los desarrolladores como de la comunidad. El proyecto RoboCup cuenta con competiciones oficiales anuales desde 1997 y cada año entran nuevos participantes que se basan en implementaciones de equipos de años anteriores.
- Que presente una implementación completa, en este caso diseñada para la simulación. El Soccer Server cuenta con las herramientas necesarias para implementar el entorno de forma completa, proporcionando al desarrollador una base sobre la que empezar a trabajar.
- Que esté formada por varios componentes diferentes, incluyendo un ejemplo de cliente, módulos de entrenador y herramientas para visualización de resultados. Así mismo el módulo de entrenador debe estar disponible para crear experimentos. El Soccer Server cuenta con módulos específicos para cada una de estas necesidades. Dichos módulos serán explicados en Anexo B.
- Que disponga de herramientas de colección y análisis de datos para realizar las mediciones de los resultados de los enfrentamientos entre equipos de la forma más rigurosa posible. Todos los partidos ejecutados en el *Soccer Server* son registrados en forma de *logs*, pudiendo ser visualizados más adelante.
- Que aporte usabilidad, robustez y escalabilidad para aumentar de forma progresiva la complejidad. La RoboCup cuenta con una documentación adecuada disponible al desarrollador. Así mismo, el sistema ha ido evolucionando buscando la mayor robustez posible, la cual se consigue definiendo un entorno lo más concreto posible con unas reglas de juego muy bien definidas. La escalabilidad viene dada por las diferentes ligas existentes dependiendo del tamaño de los robots, o del entorno utilizado en competiciones de simulación.

- Que disponga de la posibilidad de inducir fallos intencionados (como deshabilitar un agente, por ejemplo). El *Soccer Server* dispone de la posibilidad de introducir agentes de tipo entrenador que tiene total libertad para modificar el entorno y manejar a los agentes participantes.

Otras características con las que cuenta y que facilitan su utilización en la comunidad investigadora son las siguientes:

- Ligereza: requiere un consumo bajo de recursos, con lo que puede utilizarse en cualquier ordenador.
- Multiplataforma: soporta Linux, SunOS, Solaris, IRIS y OSF/1. Existen versiones para Windows de la mayor parte de las herramientas, aunque no están actualizadas. El hecho de que sea multiplataforma y que únicamente precise de herramientas comunes como librerías, compiladores de C++ y un servidor gráfico, lo hace muy accesible.
- Utilización de protocolos de comunicación estándar: esto hace que cualquier lenguaje de programación sea posible, ya que salvo excepciones el protocolo *UDP/IP* está soportado por cada uno de ellos.

2.5.2 FUNCIONAMIENTO DEL SOCCER SERVER

El *Soccer Server* está diseñado de tal forma que permita la recreación de un partido de fútbol entre dos equipos rivales. Para ello provee un campo de fútbol virtual sobre el que se simulan los movimientos de los jugadores y de la pelota. Es importante señalar que todos los jugadores tienen las mismas habilidades (utilizan los mismos sensores, y sus atributos son iguales en cuanto a fuerza y precisión a la hora de golpear la pelota, o energía disponible), así que la diferencia de rendimiento entre equipos deriva únicamente del uso efectivo de los comandos de control y de la información percibida, y especialmente de la habilidad de crear comportamientos colaborativos entre múltiples agentes.

Para proporcionar un sistema de comunicación entre el agente futbolista y el servidor, se decidió instaurar una serie de conexiones a través una red que utiliza *sockets* sobre el protocolo *UDP/IP*. Esta comunicación es necesaria para que el agente envíe las acciones que desea realizar, y para recibir del servidor la información obtenida a través de los sensores del agente.

Los tres módulos del servidor son los siguientes:

1. **Módulo de simulador de campo**, que crea el mundo virtual del campo de fútbol, y calcula los movimientos de los objetos, calculando las colisiones.
2. **Módulo de árbitro**, para asegurar que las normas de juego son cumplidas.
3. **Módulo de tablero de mensajes**, que controla la comunicación entre los programas del cliente.

La Figura 6, basada en el manual de usuario de la RoboCup, muestra la representación virtual del terreno de juego, proporcionando un área delimitada por una serie de balizas o marcadores que son utilizados por los agentes para obtener una noción aproximada de posición global.

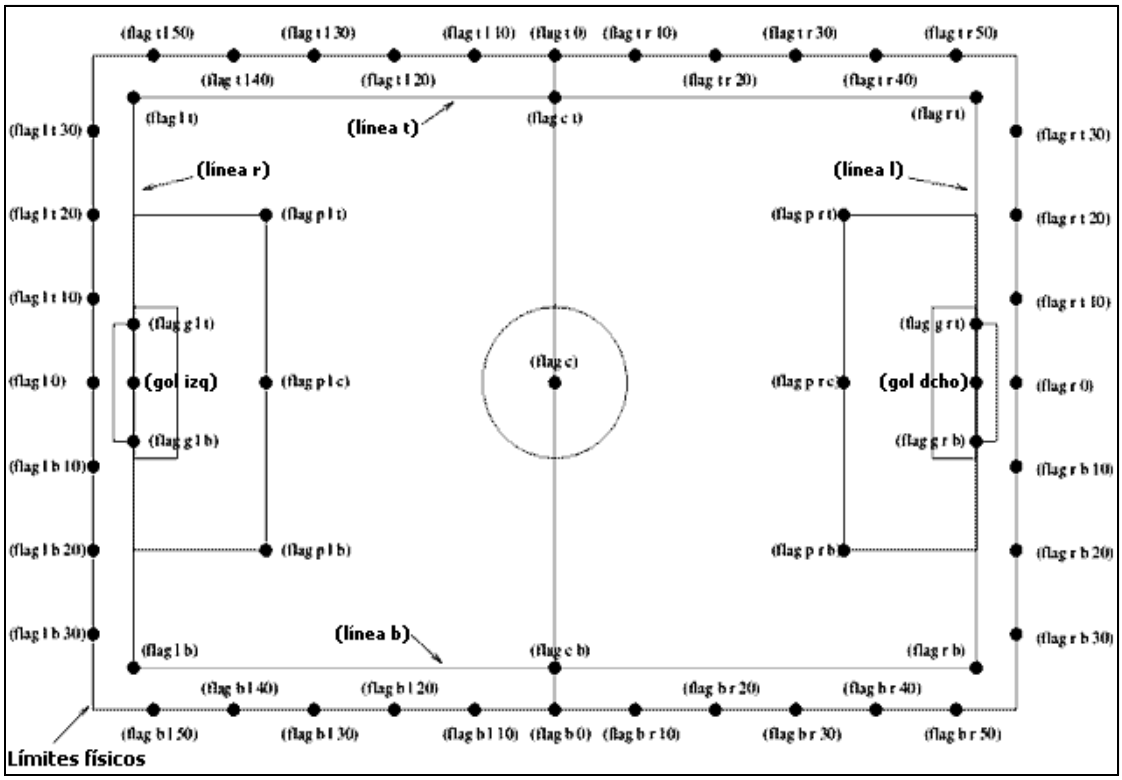


Figura 6: Simulador del campo[5]

La Figura 7 muestra el sistema de comunicación bidireccional entre los diferentes agentes y el *Soccer Server*, siempre a través de *sockets*. Es muy importante considerar que cada agente es autónomo, es decir, cada cliente controla a un único jugador.

[5] Simulador del campo, imagen basada en el manual de usuario del RoboCup *Soccer Server* [15]

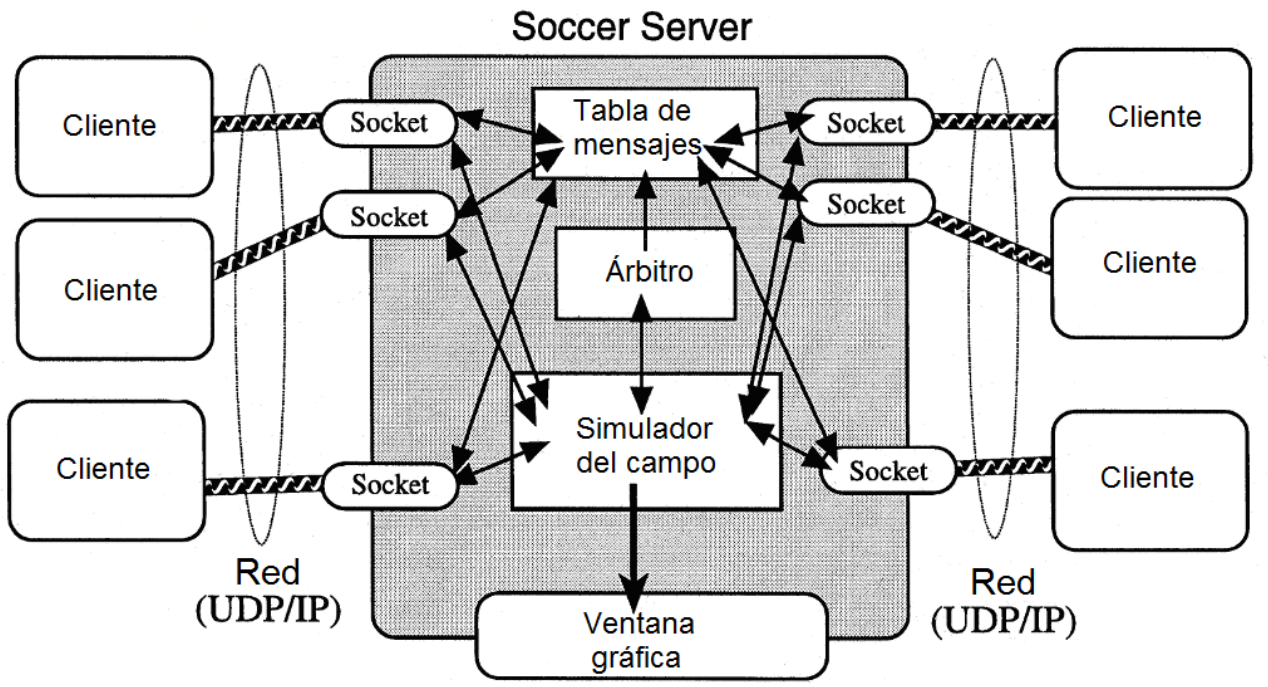


Figura 7: Vista general del *Soccer Server*[6]

La comunicación entre el servidor y cada cliente se realiza a través de cadenas *ASCII* cuyo formato ha sido previamente establecido. Al basarse la comunicación únicamente en el envío de tramas de datos, se posibilita que el agente esté programado en cualquier lenguaje de programación, ya que el único requisito es que sea capaz de manejar *sockets UDP* y tener funciones de manejo de cadenas, lo cual es aplicable en la inmensa mayoría de los lenguajes de programación existentes.

El protocolo de comunicación consiste en:

- **Comandos de control**: los mensajes que manda un cliente al servidor para controlar las acciones del jugador asociado al cliente. Los comandos básicos son girar (comando "*turn*"), avanzar (comando "*dash*") y chutar (comando "*kick*"). La comunicación entre compañeros de equipo, en caso de que sea permitida, se produce a través del comando hablar ("*say*"), y existe un comando único para el cliente portero que es intentar atrapar la pelota (comando "*catch*").
- **Información percibida**: mensajes enviados del servidor al cliente describiendo la situación actual del juego desde el punto de vista del cliente. Hay tres tipos de información: visual (comando "*see*"), auditiva (comando "*hear*") y percibida (comando "*sense_body*").

Estos comandos enviados entre el servidor y los agentes son procesados dentro de una estructura de simulación por ciclos, con lo cual el procesamiento no es continuo sino que se ajusta a unos

[6] Vista general del *Soccer Server*, imagen basada en Noda y Stone [16]

períodos específicos. Concretamente las instrucciones de control se procesan cada 100 ms. y la información es percibida por el agente en rangos de entre 37 y 300 ms., variables según la calidad de la información percibida.

Además del servidor y los agentes jugadores, el servidor puede proveer de un *socket* extra para un entrenador, que será un agente con habilidades especiales, entre ellas tener una visión global del conjunto y poder dirigir aspectos del juego como mover objetos o enviar mensajes al resto de agentes. El entrenador es muy importante a la hora de optimizar equipos, puesto que permite generar simulaciones con condiciones concretas y almacenar los resultados de éstas. Con el paso de las versiones se ha incluido un modo de entrenador online limitado que puede participar una vez comenzado el partido, pudiendo comunicarse con los clientes para transmitirles órdenes de estrategia o para darles consejos tácticos.

2.6 EQUIPO CMUNITED 99

En el presente punto se explica el funcionamiento del agente jugador, también llamado cliente del *Soccer Server*. Este proyecto ha tomado como referencia para el desarrollo al agente CMUnited 99 [37].

Se ha optado por utilizar un cliente ya disponible y no crear uno nuevo para poder trabajar directamente en las nuevas funcionalidades que se desean añadir, puesto que habría que desarrollar una base que permitiera al cliente manejar la comunicación con el servidor, además de elaborar un modelo del mundo actualizable, y de combinar las primitivas de acción a bajo nivel en habilidades utilizables por el propio cliente.

El cliente CMUnited 99 ha sido programado por el equipo de desarrollo de la Universidad Carnegie Mellon en la ciudad de Pittsburg, Pennsylvania, en Estados Unidos. Ha sido seleccionado por su satisfactoria experiencia en ediciones anteriores de la RoboCup, puesto que este equipo ganó las ediciones de los años 1998 y 1999. Además varios desarrolladores se basaron en CMUnited para realizar sus propios clientes. Por ejemplo, el equipo que quedó tercero en el año 1999 era una adaptación parcial del cliente CMUnited del año 1998, y los tres primeros equipos de la RoboCup 2000 estaban basados en el código fuente del cliente CMUnited 99. Concretamente el tercer clasificado fue el equipo ATT-CMUnited-2000, la evolución que presentó la propia Universidad Carnegie Mellon. Después de ese año se decidió evolucionar utilizando como base el código liberado de otros agentes.

2.6.1 ARQUITECTURA DEL AGENTE

Como cualquier agente, el cliente CMUnited 99 tiene habilidades de percepción del medio, de racionalización para decidir el comportamiento a seguir, de comunicación con el resto de los agentes, y de actuación para ejecutar las acciones deseadas.

En la base de los agentes CMUnited 99 está lo que se llama “acuerdo de vestuario”, como lo definen Noda y Stone [16]. Basado en la premisa de que los agentes se pueden encontrar periódicamente en un entorno seguro, con comunicación disponible, el acuerdo de vestuario define cómo deben actuar cuando se encuentren los agentes en un entorno con baja calidad de comunicación, en el cual el tiempo es crítico, y en el que los adversarios también participan.

Los agentes individuales pueden tomar acuerdos de vestuario y responder al entorno mientras están actuando de forma individual, esto es, percibiendo el entorno, razonando y seleccionando sus acciones, y actuando en dicho entorno. En las oportunidades de sincronización con el resto del equipo, el conjunto además hace un acuerdo de vestuario para ser usado por todos los agentes durante los períodos de comunicación limitados. La Figura 8 muestra el modelo de entrada/salida de esta arquitectura.

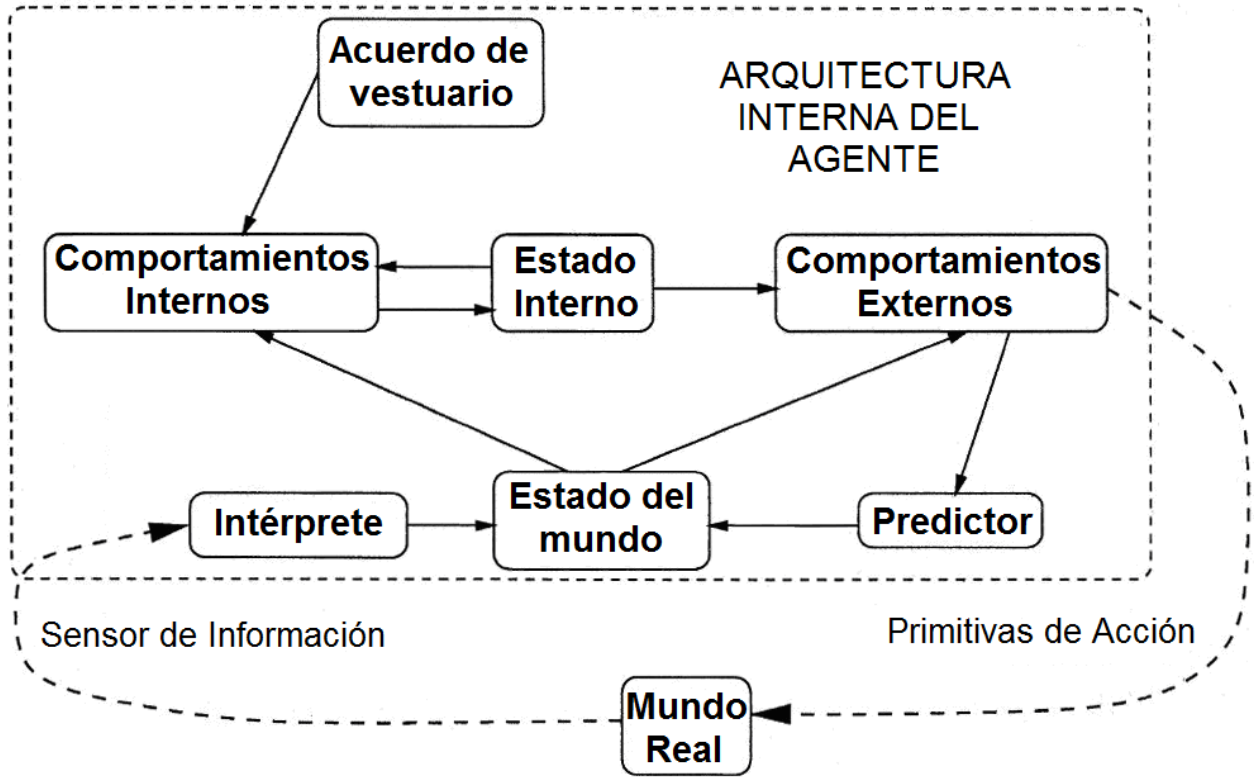


Figura 8: Modelo funcional de E/S de la arquitectura del agente CMUnited 99[7]

El agente guarda registros de tres tipos de estados: el estado del mundo, el acuerdo de vestuario, y el estado interno. Además el agente tiene dos tipos de comportamientos, internos y externos.

- **El estado del mundo** refleja la concepción del agente del mundo real, tanto a través de sus sensores como con la predicción de los efectos de sus acciones. Este estado es actualizado como resultado de la interpretación de la información, para lo cual se ha de guardar un registro del estado del mundo en los instantes de tiempo anteriores. También

[7] Modelo funcional de E/S de la arquitectura del agente CMUnited 99, imagen basada en el manual de usuario del RoboCup *Soccer Server* [15]

puede ser actualizado según los efectos que presumiblemente tendrá el módulo de comportamiento externo de las acciones elegidas. El estado del mundo es accesible directamente tanto por los comportamientos internos como externos.

- **El acuerdo de vestuario** se define por el equipo cuando tienen la posibilidad de sincronizarse privadamente. Define la estructura de cooperación del equipo y los protocolos de comunicación entre agentes, si los hubiera. El acuerdo de vestuario es accesible únicamente a los comportamientos internos.
- **El estado interno** almacena las variables internas del agente. Puede reflejar el estado actual o estados previos del mundo, configuración que puede ser especificada en el acuerdo de vestuario. El agente actualiza su estado interno a través de los comportamientos internos.
- **Los comportamientos internos** actualizan el estado interno del agente basado en su propio estado interno, el estado del mundo, y el acuerdo de vestuario.
- **Los comportamientos externos** referencian al mundo y los estados internos, y seleccionan las acciones que se envían a los actuadores, es decir, las acciones a ejecutar, alterando el estado del mundo. Los comportamientos externos consideran únicamente el mundo y los estados internos, sin tener en cuenta el acuerdo de vestuario.

2.6.2 MODELADO DEL MUNDO

Al basarse las decisiones a efectuar en el modelo del mundo, es importante que el agente cree un modelo del mundo lo más preciso posible. Las consecuencias de las acciones del agente no son determinísticas, y por tanto la evolución del mundo será una incógnita para el agente. Para adaptarse lo mejor posible al entorno, el agente ha de estar recopilando información en tiempo real, y con ella creará el modelo en el que se basarán sus predicciones.

En el entorno cerrado de la RoboCup siempre se van a tratar con los mismos elementos, que se pueden diferenciar en objetos estáticos, cuya posición va a permanecer invariable en toda la simulación, y en objetos dinámicos, en constante movimiento. Los objetos estáticos, como son las porterías y los marcadores del campo, son utilizados por el agente para posicionarse a sí mismo en el campo. Los objetos móviles requieren de atención continua, y son el propio agente, la bola, y los otros veintiún jugadores (de los cuales diez son compañeros y el resto rivales). Para diferenciar los distintos jugadores se utilizan variables de equipo y números de uniforme, que son conocidas de antemano.

Los objetos móviles son almacenados con valores de confianza entre [0,1], indicando la seguridad con la que su localización es conocida. Estos valores son necesarios debido a que ningún objeto es visto de forma completamente consistente, ya que la información percibida es subjetiva y además puede tener ruido.

Las variables asociadas a cada tipo de objeto con las siguientes:

Tipo de objeto:

- Objeto estático
  - Posición en coordenadas globales (*x*, *y*)
  - Confianza acerca de la precisión de dichas coordenadas [0,1]
- Objeto dinámico
  - Balón:
    - Posición en coordenadas globales (*x*, *y*)
    - Confianza acerca de la precisión de dichas coordenadas [0,1]
    - Coordenadas de velocidad globales (*dx*, *dy*)
    - Confianza acerca de la precisión de dichas coordenadas [0,1]
  - Jugador:
    - Posición en coordenadas globales (*x*, *y*)
    - Confianza acerca de la precisión de dichas coordenadas [0,1]
    - Coordenadas de velocidad globales (*dx*, *dy*)
    - Confianza acerca de la precisión de sus coordenadas [0,1]
    - Equipo
    - Número de uniforme
    - Ángulo de orientación global θ
    - Confianza acerca de la precisión del ángulo [0,1]

Para actualizar este modelo del mundo, se utiliza la información visual, la información auditiva, la información percibida, y las predicciones realizadas en base a acciones tomadas anteriormente. Esta actualización es constante y necesaria debido a que las condiciones cambian en cada ciclo de ejecución.

2.6.3 HABILIDADES DEL AGENTE

Una vez que el agente ha determinado el estado del mundo de la forma más precisa posible, el siguiente paso es enviar al servidor la acción concreta que va a ejecutar. Al hacerlo puede seleccionar entre varias habilidades a bajo nivel que proveen las primitivas básicas (avanzar con una fuerza determinada, girar un determinado ángulo, chutar la pelota con una determinada fuerza), dando salida a los siguientes comandos de movimientos básicos:

- *Golpear la pelota*: chutar la bola para avanzar hacia la portería contraria, pasar a un compañero, o disparar con la intención de anotar un gol.
- *Driblar*: regatear en posesión de la pelota para evitar que el rival se haga con ella.
- *Interceptar la pelota*: interponerse en la trayectoria del balón para hacerse con la posesión.
- *Parar la pelota*: en el caso del portero, coger la pelota para evitar que se produzca un gol o que la pelota salga fuera del terreno de juego.

- *Defender*: interponerse el jugador de campo entre los atacantes y la portería contraria para dificultar el avance del equipo rival.
- *Despejar*: chutar la pelota con fuerza hacia fuera del campo para evitar el peligro rival.

Estas acciones se lanzarán cuando el efecto que tenga su ejecución sea el óptimo para el agente. Además todas estas habilidades tienen unos efectos predecibles, con lo que se podrá determinar con relativa precisión el estado del mundo después de realizar dicha acción. Aunque las habilidades sean predecibles, el agente ejecuta únicamente una acción durante cada ciclo. Cuando llega el momento de actuar de nuevo, la situación es completamente reevaluada. Esto implica que, si el estado del mundo es muy similar a la configuración anticipada, el agente actuará de forma similar a la que predijo en ciclos anteriores. Sin embargo, si el mundo es significativamente diferente, el agente tendrá que llegar a una nueva secuencia de acciones completamente diferente, en la que de nuevo ejecutará únicamente el primer paso de la nueva secuencia.

2.7 MODELADO DE AGENTES BASADO EN LA OBSERVACIÓN

En Ledezma [3], el trabajo en el que se basa este proyecto, se propone la utilización de técnicas de aprendizaje automático con el propósito de llevar a cabo la tarea del modelado del comportamiento de los agentes basándose en técnicas IOAM (*Input-Output Agent Modeling*). Estas técnicas se basan en la observación del comportamiento de un agente con la intención de llevar a cabo la tarea de modelado basándose en la relación existente entre la entrada y la salida de dicho agente.

Con dicho propósito se definió el modelo MABO (Modelado de Agentes Basado en la Observación), que se utiliza en sistemas de alta complejidad en el que el agente no tiene acceso directo a las entradas y salidas del agente a modelar, es decir, no tiene un conocimiento de antemano de la estructura interna del agente a modelar.

El MABO parte de la observación y recolección de datos, y está formado por tres módulos diferenciados, el MEA (Módulo de Etiquetado de Acciones), el MCM (Módulo de Construcción del Modelo) y el MRA (Módulo de Razonamiento). La Figura 9 muestra el marco general de disposición de los elementos del MABO.

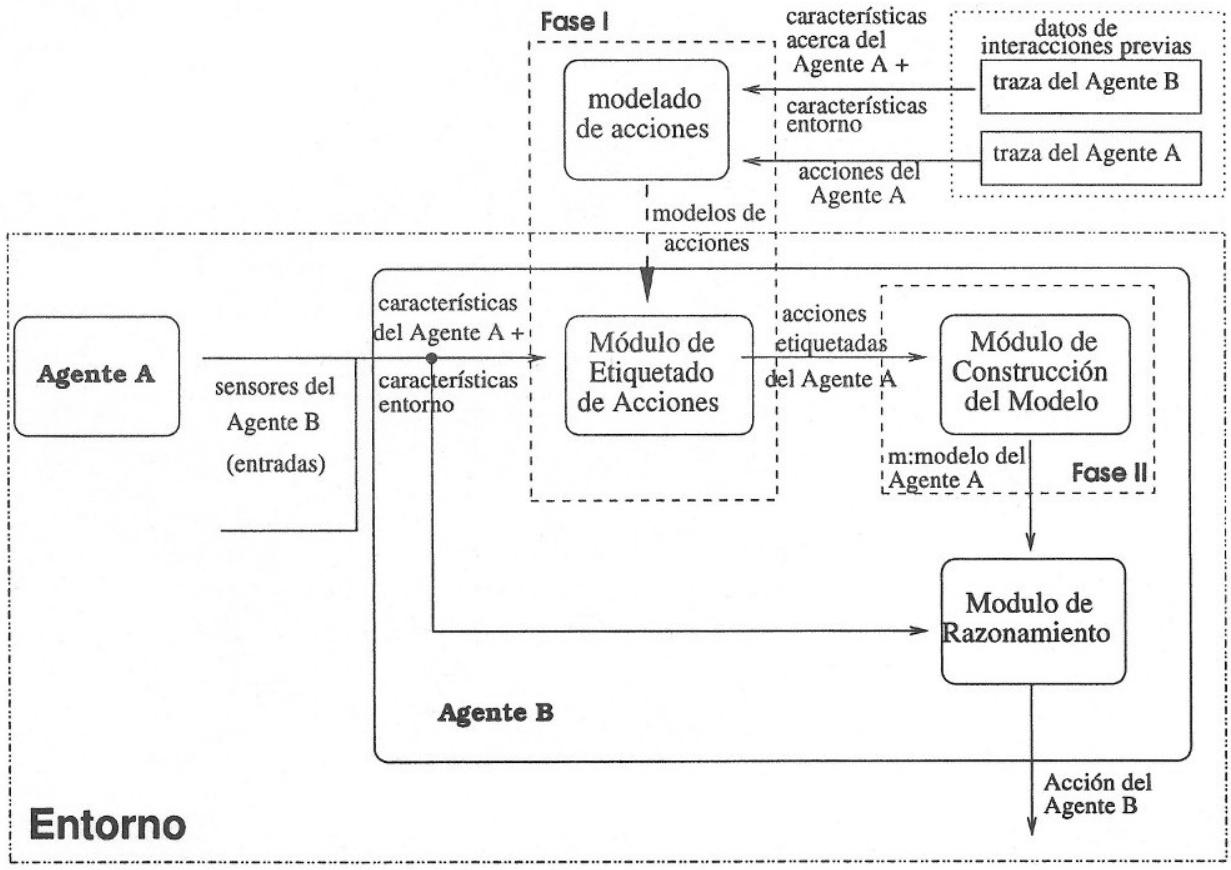


Figura 9: Creación del Módulo de Modelado de Agentes Basado en la Observación[8]

El MEA es el encargado de etiquetar las acciones llevadas a cabo por el agente objeto de estudio (en adelante agente A) en las observaciones de éste realizadas por el agente etiquetador (en adelante agente B). Así pues, se necesitan instancias con datos del agente A y de la acción que toma, tal y como el agente etiquetador lo percibe.

[8] Creación del Módulo de Modelado de Agentes Basado en la Observación, imagen extraída de Ledezma [3]

La Figura 10 muestra una descripción de cómo se genera del Módulo de Etiquetado de Acciones.

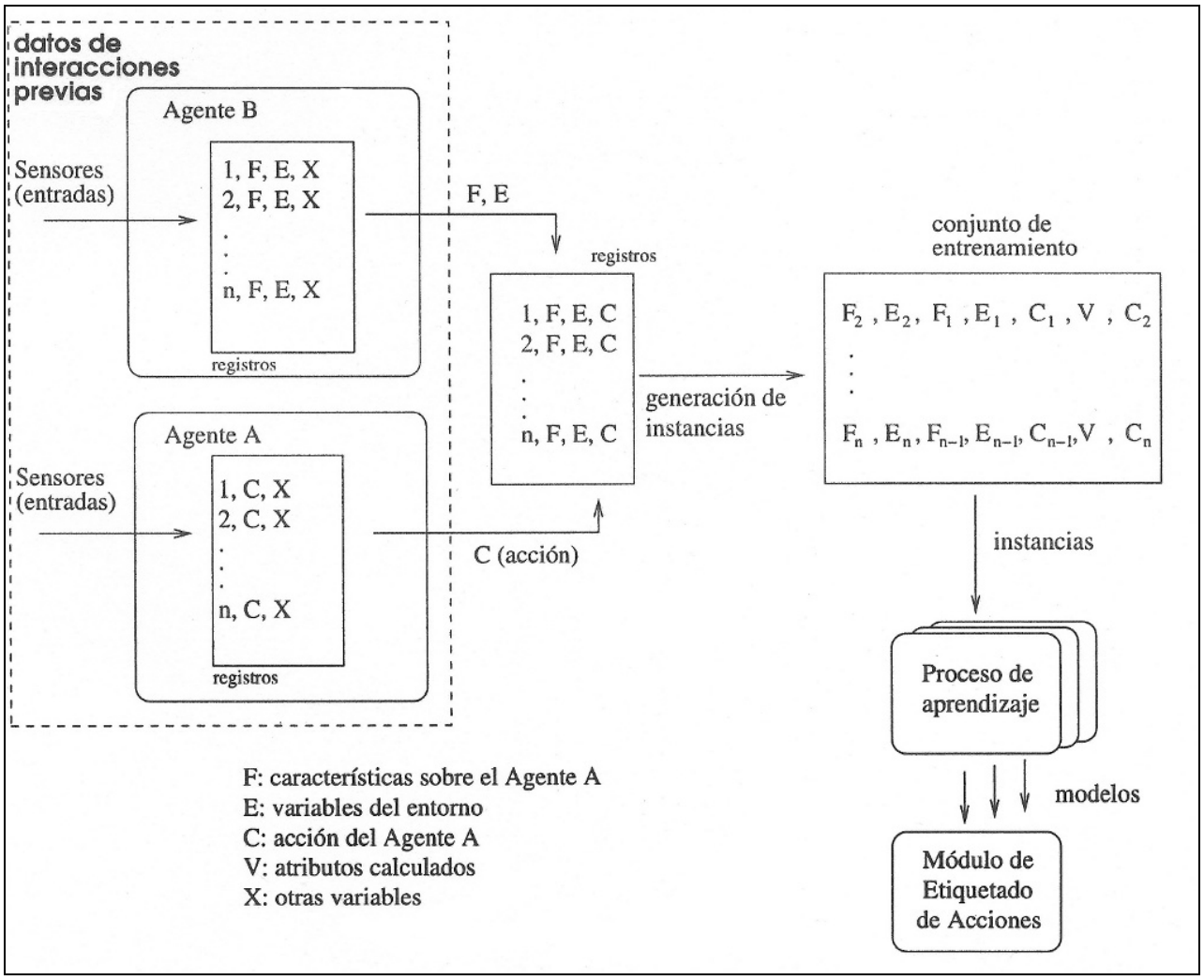


Figura 10: Creación del Módulo de Etiquetado de Acciones[9]

Los pasos llevados a cabo para la construcción del MEA son los siguientes:

- El agente A y el agente B interactúan un número determinado de veces. En cada instante el agente etiquetador registran variables relacionadas con el agente A, variables del entorno obtenidas por él mismo, y las propias acciones del agente A, con el objetivo de obtener una traza del comportamiento del agente A desde el punto de vista del agente B.
- Cada registro en la traza está compuesto por un grupo de atributos relacionados con el agente A (F), variables de entorno (E), y la acción llevada a cabo por el agente A (C). Este formato será de utilidad más adelante al combinar varias trazas consecutivas.
- El objetivo es, a través de este conjunto de datos, obtener una traza de atributos con la mayor información posible basándose en la comparación de las diferencias de variables

[9] Creación del Módulo de Etiquetado de Acciones, imagen extraída de Ledezma [3]

entre dos instantes de tiempo contiguos, asociándolo a una acción determinada por parte del agente A. De este modo se realiza una asociación atributos percibidos – acción realizada que será la base para realizar la clasificación.

- Estas trazas generadas se utilizan para construir clasificadores con el fin de etiquetar la acción llevada a cabo por el agente A.

Por otra parte, una vez construido el MEA e incorporado en el agente B, el siguiente paso consiste en construir el Módulo de Construcción del Modelo o MCM, es decir, crear un modelo capaz de predecir el comportamiento del agente A basado en las observaciones realizadas desde el punto de vista del agente B. Los pasos efectuados son los siguientes:

- MEA se incorpora a la arquitectura del agente B, para poder inferir las acciones del agente a estudiar.
- Al igual que en el MEA, ambos agente interactúan durante un número determinado de veces, con el objetivo de producir una traza del comportamiento del agente A.
- Cada ejemplo en la traza está compuesto por los mismos elementos que en el MEA, salvo que la acción llevada a cabo por el agente A es la acción etiquetada por el MEA.
- Para realizar la predicción de la acción en un instante determinado se necesita información del agente A en instantes anteriores, así que la traza obtenida se compone de la información observada en un número concreto de instantes de tiempo.
- Al igual que en el MEA, con estas trazas generadas se construyen clasificadores capaces de predecir la acción llevada a cabo por el agente A en un instante de tiempo dado.

Por último, ya que la predicción en sí no sirve para nada, se ha de construir un Módulo de Razonamiento o MRA, que es utilizado a la hora de tomar la decisión sobre qué acción llevar a cabo, e incorporarlo en el agente B para que el agente modelador se pueda anticipar y reaccionar a las acciones del oponente. Para la construcción de dicho módulo también se utilizan técnicas de aprendizaje automático.

En resumen, el MABO consiste, partiendo de los datos percibidos por un agente, en la creación de un modelo lo más preciso posible del comportamiento de un agente. Para ello se ha de construir un etiquetador de las acciones del agente objeto de estudio a través del aprendizaje. Dicho etiquetador se incorpora al agente, y a través de la observación se obtiene información para crear un modelo para predecir las acciones del agente objeto de estudio. Una vez obtenido el modelo, se integra al agente para sacar provecho de él en el entorno.

3 PLANTEAMIENTO Y DESARROLLO DEL PROYECTO

3.1 INTRODUCCIÓN

En este punto se va a explicar el núcleo del proyecto, es decir, se va llevar a cabo una presentación en profundidad del trabajo realizado. En las siguientes líneas se detallará por tanto la arquitectura del entorno, con todos los programas de los que consta y los procesos que se van enlazando para llegar al objetivo de construir un modelo del agente en base a los datos observados.

Para comprender este punto, es importante remitirse al doble objetivo del proyecto

1. La primera parte consiste en la optimización de la implementación del modelo de Modelado de Agentes Basado en la Observación MABO en el entorno de la RoboCup a través del agente CMUnited 99. En esta parte se discutirá la implementación práctica del modelo teórico en el agente, junto con las utilidades desarrolladas y las mejoras implementadas.
2. La segunda parte consiste en verificar los resultados obtenidos al poner a competir al agente obtenido en la primera parte junto a otros equipos. Esta sección confirmará que las utilidades desarrolladas cumplen su cometido y que el aprendizaje se efectúa independientemente del agente objeto de estudio.

3.2 OPTIMIZACIÓN DEL PROCESO DE MODELADO

Este proyecto toma la aplicación del MABO realizada en Ledezma [3] como base y plantea una optimización a través de una serie de utilidades. Como se ha visto en el punto 2.7, dicho modelo consta de dos partes fundamentales: el proceso de creación del etiquetador, y el proceso de construcción del modelo (de ahora en adelante, proceso del modelador).

Como ya se ha comentado, el MABO se basa en la observación y recolección de datos. Estos datos a percibir van a ser obtenidos a través del *Soccer Server*, la infraestructura base de la liga de simulación 2D de la RoboCup. En el siguiente punto se trata todo lo relacionado con la obtención de datos.

3.2.1 RECOLECCIÓN DE DATOS BASADA EN LA OBSERVACIÓN

En el entorno de la RoboCup la única manera de obtener datos útiles es mediante la simulación de varios partidos. El agente objetivo y el rival se enfrentan en un entorno preparado; habitualmente en los partidos intervienen once agentes contra otros once pero en la configuración de las pruebas se ha optado por limitar este escenario y que únicamente intervenga el delantero (agente al que se le incorporará el modelo) contra el portero (agente que será observado).

Para realizar la simulación se utiliza un agente especial llamado *Trainer*, que se encarga de generar unas condiciones particulares en el que los agentes puedan competir. Genera partidos con unas condiciones muy concretas, situando los elementos dinámicos del juego en posiciones concretas al inicio de cada iteración, por lo que permite crear un escenario propicio para el estudio. Así pues, el *Trainer* es un agente de apoyo, a diferencia del agente delantero y el portero que tienen otra tarea mucho más definida, meter goles o evitarlos, respectivamente.

3.2.1.1 ENTORNO DE PRUEBAS

El *Trainer* se va a encargar de lanzar un número concreto de ejecuciones (en este caso 30 ejecuciones, pero que se repetirán 100 veces para obtener resultados lo más arbitrariamente posible), en las que el delantero saldrá desde un punto aleatorio del campo, tomará el balón, y con él en su posesión llegará hasta el portero, y procurará meter gol chutando a puerta el balón. A su vez, el portero intentará por todos sus medios detener la pelota. Cada vez que el balón entre a gol, o bien sea detenido por el portero, o bien se salga de los límites del campo, acabará la jugada. Para realizarlo se ejecutan los diferentes agentes sobre el servidor, que es el programa base sobre el que se realiza la simulación y que se encarga de manejar las llamadas entre los agentes y habilitar el juego. Así pues se cuenta con los siguientes programas:

- *Soccer Server*: programa base sobre el que corren las ejecuciones. En un símil con el juego de fútbol real, representa el terreno de juego y el árbitro. Simplificando sus funciones, define las posiciones de los objetos estáticos, y además define las reglas del juego y se encarga de controlar el número de goles de uno y otro equipo.
- *Trainer*: agente supervisor que coloca a los jugadores después de cada jugada. En comparación al juego de fútbol real, representa a un árbitro auxiliar que después de cada jugada se encarga de poner el balón en el centro del campo y se asegura de que cada jugador esté donde le corresponde antes de empezar una nueva ejecución.
- *Delantero*: agente observador, su objetivo es marcar goles.
- *Portero*: agente del que se quiere modelar su comportamiento para adaptarse a él, su objetivo es detener los goles que intenta marcar el delantero.

3.2.1.2 OBTENCIÓN DE LA INFORMACIÓN DEL AGENTE RIVAL

La información del comportamiento del agente rival se obtiene a través de dos fuentes: los datos obtenidos por el propio agente gracias a sus sensores y, en el caso de estar disponibles, los datos de las acciones llevadas a cabo por el agente rival, que se obtienen a través del *Soccer Server*.

La primera fuente corresponde a los sensores que tiene el propio agente objetivo, que se refiere a los atributos que puede percibir el agente. Dichos atributos son mostrados en la
Tabla 1:

NOMBRE	DESCRIPCIÓN	TIPO
SeeOpponent	¿puedo ver al oponente?	Sí/no
OpponentNumber	número del oponente	Numérico
BallKickableForOpponent	¿puede el oponente disparar el balón?	Sí/no
CanFaceOpponentWithNeck	¿Puedo encarar al oponente girando el cuello?	Sí/no
CanSeeOpponentWithNeck	¿Puedo ver al oponente girando el cuello?	Sí/no
BallMoving	¿El balón se está moviendo?	Sí/no
BallKickable	¿Puedo disparar al balón?	Sí/no
OpponentPositionValid	Grado de certeza sobre la posición del oponente	Numérico
OpponentDistance	Distancia al oponente	Numérico
OpponentSpeed	Velocidad del oponente	Numérico
OpponentAngleFromBody	Ángulo del oponente desde mi cuerpo	Numérico
OpponentAngleFromNeck	Ángulo del oponente desde mi cuello	Numérico
BallPositionValid	Grado de certeza sobre la posición del balón	Numérico
BallSpeed	Velocidad del balón	Numérico
BallDistance	Distancia del balón	Numérico
BallAngleFromBody	Ángulo del balón desde mi cuerpo	Numérico
BallAngleFromNeck	Ángulo del balón desde mi cuello	Numérico
MyBodyAngle	Ángulo de mi cuerpo	Numérico
MySpeed	Mi velocidad	Numérico
MyAction	Mi acción	Turn, dash, kick
MyActionAngle	El ángulo asociado a mi acción	Numérico
MyActionPower	La fuerza asociada a mi acción	Numérico

Tabla 1: Atributos sensados por el agente

Todos esos atributos se guardan para el instante *t* y para el instante *t-1*, es decir, un instante determinado y el instante inmediatamente anterior.

Para obtener una relación de los atributos calculados se puede observar la
Tabla 2:

NOMBRE	DESCRIPCIÓN
DIF_BKFO	diferencia en dos instantes de tiempo del atributo BallKickableForOpponent
DIF_CFOWN	diferencia en dos instantes de tiempo del atributo CanFaceOpponentWithNeck
DIF_CSOWN	diferencia en dos instantes de tiempo del atributo CanSeeOpponentWithNeck
DIF_BM	diferencia en dos instantes de tiempo del atributo BallMoving
DIF_BK	diferencia en dos instantes de tiempo del atributo BallKickable
DIF_OX	diferencia en dos instantes de tiempo de la coordenada X del oponente
DIF_OY	diferencia en dos instantes de tiempo de la coordenada Y del oponente
DESP_O	desplazamiento del oponente de un instante de tiempo a otro (distancia euclídea)
DIF_OD	diferencia en dos instantes de tiempo del atributo OpponentDistance

NOMBRE	DESCRIPCIÓN
DIF_OS	diferencia en dos instantes de tiempo del atributo OpponentSpeed
DIF_OAFB	diferencia en dos instantes de tiempo del atributo OpponentAngleFromBody
DIF_OAFN	diferencia en dos instantes de tiempo del atributo OpponentAngleFromNeck
DIF_BX	diferencia en dos instantes de tiempo de la coordenada X del balón
DIF_BY	diferencia en dos instantes de tiempo de la coordenada Y del balón
DESP_Ball	desplazamiento del balón de un instante a otro (distancia euclídea)
DIF_BS	diferencia en dos instantes de tiempo del atributo BallSpeed
DIF_BD	diferencia en dos instantes de tiempo del atributo BallDistance
DIF_BAFB	diferencia en dos instantes de tiempo del atributo BallAngleFromBody
DIF_BAF	diferencia en dos instantes de tiempo del atributo BallAngleFromNeck
DIF_MyX	diferencia en dos instantes de tiempo de mi coordenada X
DIF_MyY	diferencia en dos instantes de tiempo de mi coordenada Y
DESP_My	mi desplazamiento de un instante a otro (distancia euclídea)
DIF_MyBA	diferencia en dos instantes de tiempo del atributo MyBodyAngle
DIF_MyS	diferencia en dos instantes de tiempo del atributo MySpeed
CLASS	acción llevada a cabo por el agente a modelar en el instante t-1

Tabla 2: Atributos calculados por el agente

Este método de representar la información es totalmente necesario, ya que en este modelo de sistemas multiagente la información siempre es parcial y depende del punto de vista del agente, por lo que es totalmente subjetiva. Es decir, el agente desconocerá la posición exacta del resto de elementos de la simulación, no puede saber la posición en coordenadas ni de los rivales, ni del balón, ni de la otra portería, ni siquiera de sí mismo. Para ayudar en el posicionamiento cuenta con una serie de balizas que proporciona el *Soccer Server*, cuya configuración fue mostrada en el punto 2.5.2. Lo único que puede conocer es la distancia aproximada a la que se encuentra un cierto objetivo y el ángulo que les separa. Además hay que sumar el hecho de que la información la recibe del simulador y que éste les proporciona la información con un ruido directamente proporcional a la distancia que separa el objeto del receptor de la información.

La Figura 11, basada en el manual de desarrollador del *Soccer Server,* muestra la visión que tiene el agente.

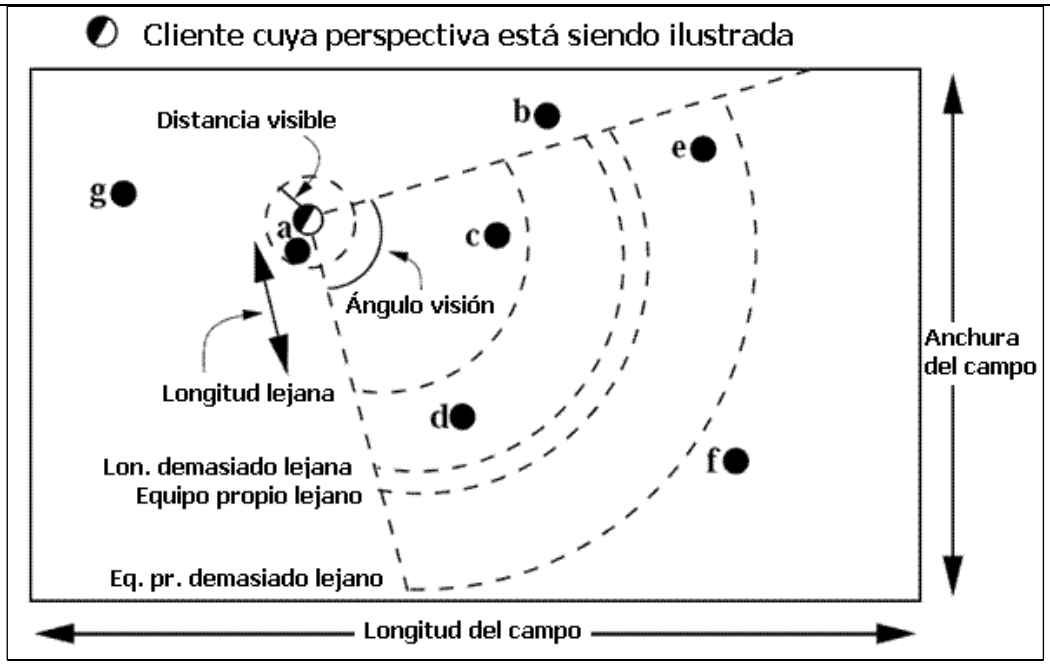


Figura 11: Visión del agente[10]

Como se ha comentado, el aprendizaje a través de la observación no maneja datos 100% precisos. Para guiar el proceso de aprendizaje, siempre en el contexto del MABO, se han incorporado los datos acerca de las acciones del agente objeto de estudio, es decir, se clasifican sus acciones. El modo de conseguir esta información consiste en utilizar los datos que envía el agente rival al *Soccer Server* para definir sus movimientos. Esta información es muy básica, y consiste en la acción que realiza el rival y los parámetros de dicha acción, es decir, si se mueve, y con qué ángulo se mueve, o si tira y la fuerza del tiro.

Las acciones que puede realizar el agente, es decir, los valores posibles de la clase asociada a cada instancia tomada, son *turn*, *kick*, *dash* y *none*, aparte de otra clase llamada *desconocida* que es asignada a una instancia de la que no se puede tener constancia real debido al ruido del dominio. Estas instancias corresponden a girar (*turn*), golpear (*kick*), avanzar (*dash*) y no hacer nada (*none*), respectivamente. Además existe la clase específica parar (*catch*) para el portero.

Así pues, siguiendo el proceso de creación del modelo de etiquetado de acciones, con estos datos se puede obtener una serie de registros con atributos percibidos por el agente, variables de entorno y la acción llevada a cabo por el agente a estudiar. La combinación de estos datos permitirá generar el etiquetador y el modelador que forman parte del MABO.

[10] Visión del agente, imagen basada en el manual de usuario del RoboCup *Soccer Server* [15]

3.2.2 GENERACIÓN DEL ETIQUETADOR

La generación del etiquetador y la generación del modelador son procesos que envuelven múltiples programas auxiliares, tienen como entrada varios ficheros de datos obtenidos a través de una experimentación previa y concluyen con un conjunto de reglas que, aplicándose a dichos datos, clasifican con mayor o menor corrección las acciones del agente rival.

La Figura 12 muestra un diagrama de flujo de datos con las fases en que se basa el proceso para la obtención del **etiquetador** y los elementos que intervienen en dicho proceso.

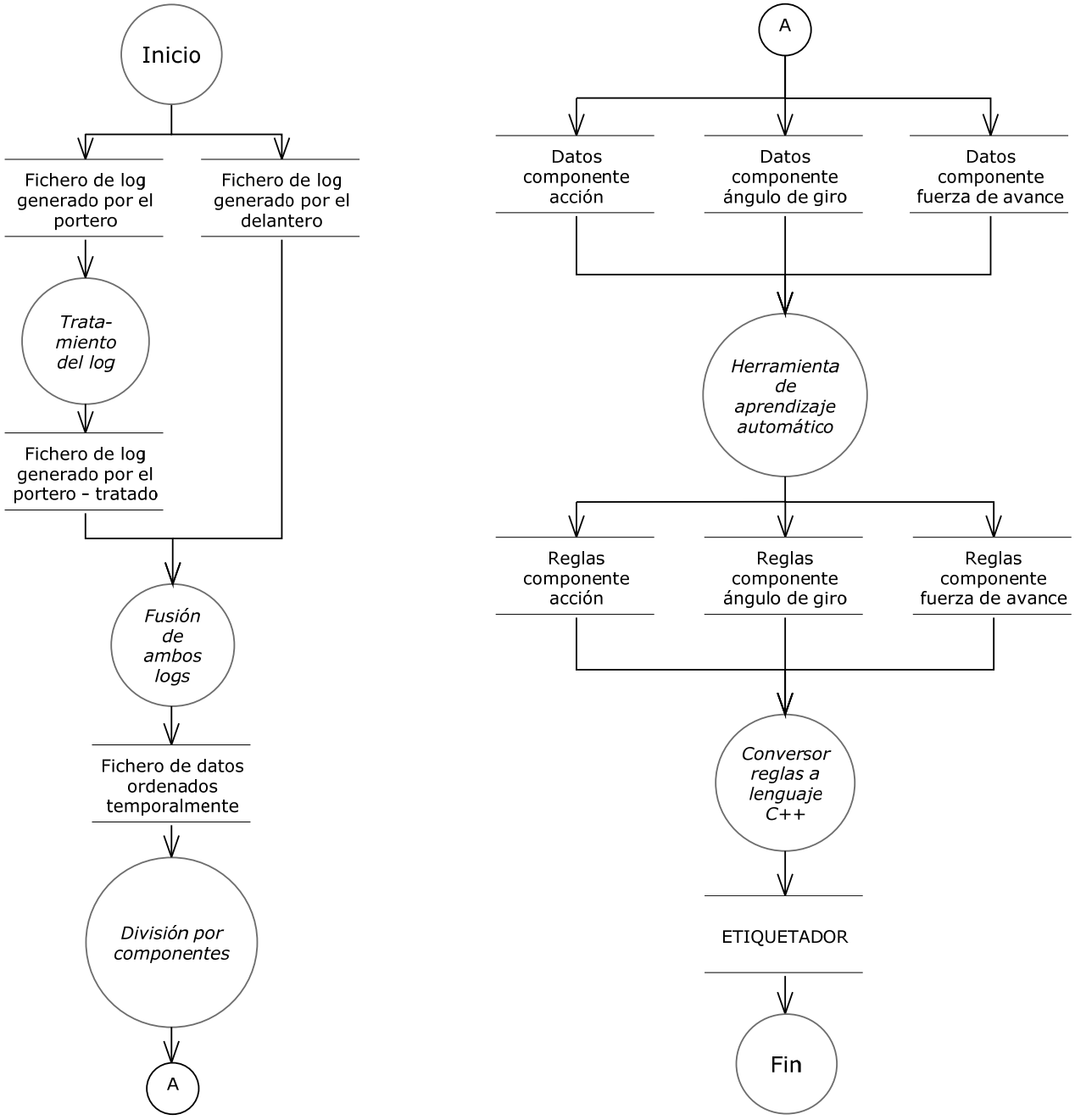


Figura 12: Esquema del proceso de creación del etiquetador

Como se ve se parte de una serie de ficheros o almacenes, representados por dos rayas paralelas, de *logs* generados por el delantero y por el portero. El fichero de *log* generado por el delantero contiene la información percibida acerca del entorno y la información que se observa del portero, mientras que el fichero de *log* del portero consiste en su lista de acciones realizadas, la cual es obtenida de los propios ficheros generados por el *Soccer Server*. Este fichero es tratado para eliminar los datos de control y la información relativa al delantero. Las herramientas intermedias o procesos, son representados por un círculo y con letra en cursiva. El siguiente paso consiste en fusionar ambos logs para ordenar los datos por tiempo. Estos datos más adelante son separados por sus distintos componentes, a través de los cuales se obtiene un etiquetador de acciones utilizando una herramienta de aprendizaje automático, en este caso la herramienta de minería de datos *Weka*. Esta herramienta aplica unos algoritmos de clasificación, generando unos ficheros de reglas, que a su vez son adaptadas al lenguaje en el que se ha programado el agente. Una vez convertidas, estas reglas se incorporan al agente para posteriormente ser utilizadas en el proceso de generación del modelador. Más adelante se explicará en profundidad cada uno de los archivos y programas envueltos hasta llegar al resultado final.

3.2.2.1 CREACIÓN DEL ETIQUETADOR: PASO A PASO

El proceso de generación del etiquetador consiste básicamente en generar datos sobre el comportamiento del portero, datos obtenidos a través de observación directa por el delantero, y extraer la información necesaria para construir unos clasificadores que más adelante serán incorporados en el agente delantero.

En el punto 3.2.2 se muestra un esquema de la construcción del etiquetador. En las líneas sucesivas se explican los diferentes pasos a seguir de manera global. Así pues los puntos a seguir son los siguientes:

- **Obtención de datos**: para crear el etiquetador hay que tomar datos de los agentes. Por tanto, el primer paso es la obtención de datos mediante la simulación de partidos en los que compiten el agente modelador contra el agente a modelar.

  Para asegurarse de que los datos obtenidos son representativos, se procede a ejecutar dos veces una simulación completa de cincuenta jugadas, esto es, cien ejecuciones completas delantero-portero. El objetivo es combinar los datos obtenidos en la dos simulaciones para obtener un conjunto de datos lo suficientemente grande como para que sea representativo y pueda servir para conseguir una correcta clasificación del comportamiento del portero. Al simular partidos el agente observador genera unos archivos de registros o *logs* donde se encuentra la información percibida. Estos *logs* son los que se toman como entrada para el proceso en la Figura 12, tanto "*Fichero de log generado por el portero*" como "*Fichero de log generado por el delantero*".

- **Tratamiento del fichero de *log* generado por el servidor**: este fichero se trata para extraer la información únicamente referida al agente rival. El *Soccer Server* guarda un registro con todos los mensajes recibidos por los agentes conectados a él, por lo que la información referida al agente observador y a las reglas de juego han de ser eliminadas, dejando únicamente un registro de acciones. Este proceso corresponde a la herramienta "*Tratamiento del log*" de la Figura 12, dando como resultado el fichero *"Fichero de log generado por el portero – tratado"*.
- **Fusión de las acciones del agente rival con los datos percibidos**: el siguiente paso es combinar tanto el fichero de *log* generado por el agente delantero con los datos percibidos como el fichero con las acciones del portero obtenido en el paso anterior. Esta unión se hace gracias a que todas las acciones registradas tienen una marca de tiempo. La herramienta utilizada en el esquema de la Figura 12 correspondiente es "*Fusión de ambos logs"*, dando como resultado el fichero "*Fichero de datos ordenados temporalmente"*.
- **División por componentes de los datos:** al fichero obtenido anteriormente se le trata con una aplicación que genera una serie de tramas de datos legibles por la herramienta de aprendizaje automático con el fin de elaborar los clasificadores, obteniendo así conjuntos de reglas que definen el comportamiento del agente rival. La herramienta utilizada en la Figura 12 corresponde a "*División por componentes*". La información se divide en tres tipos diferentes según sea el parámetro sobre el cual se clasifica:
    - información global de todas las acciones que realiza el agente a modelar. Correspondiente al fichero "*Datos componente acción*".
    - información particular para deducir el ángulo de giro en el caso de que la acción a tomar sea *turn* o girar. Correspondiente al fichero "*Datos componente ángulo de giro*".
    - información para deducir la fuerza de avance en el caso de que la acción sea *dash* o avanzar. Correspondiente al fichero "*Datos componente fuerza de avance*".
- **Obtención de reglas de clasificación para cada componente**: cada uno de los ficheros obtenidos en el punto anterior son utilizados como entrada para el proceso de minería de datos, que obtiene unos árboles de clasificación para cada componente. El proceso correspondiente en la Figura 12 es "*Herramienta de aprendizaje automático*", y los ficheros resultantes son "*Reglas componente acción*", "*Reglas componente ángulo de giro*" y "*Reglas componente fuerza de avance*".
- **Inclusión de los ficheros de reglas en el agente**: el siguiente paso es convertir esos árboles de reglas al lenguaje C++ para integrarlo en el agente. En la Figura 12 esta herramienta corresponde a "*Conversor de reglas a lenguaje C++*".

En el Anexo B: Manual de usuario, se explica en detalle el proceso.

Una vez explicados los diferentes pasos de creación del etiquetador, es necesario referenciarlo con el esquema del MABO argumentado anteriormente. En la Figura 10 se observa la creación del Módulo de Etiquetado de Acciones o MEA. Es precisamente la creación del MEA la que se lleva a cabo en este apartado, y como se puede observar se relacionan perfectamente ambos esquemas. La Figura 13, tomando como punto de partida la Figura 10,**¡Error! No se encuentra el origen de la referencia.** muestra estas relaciones.

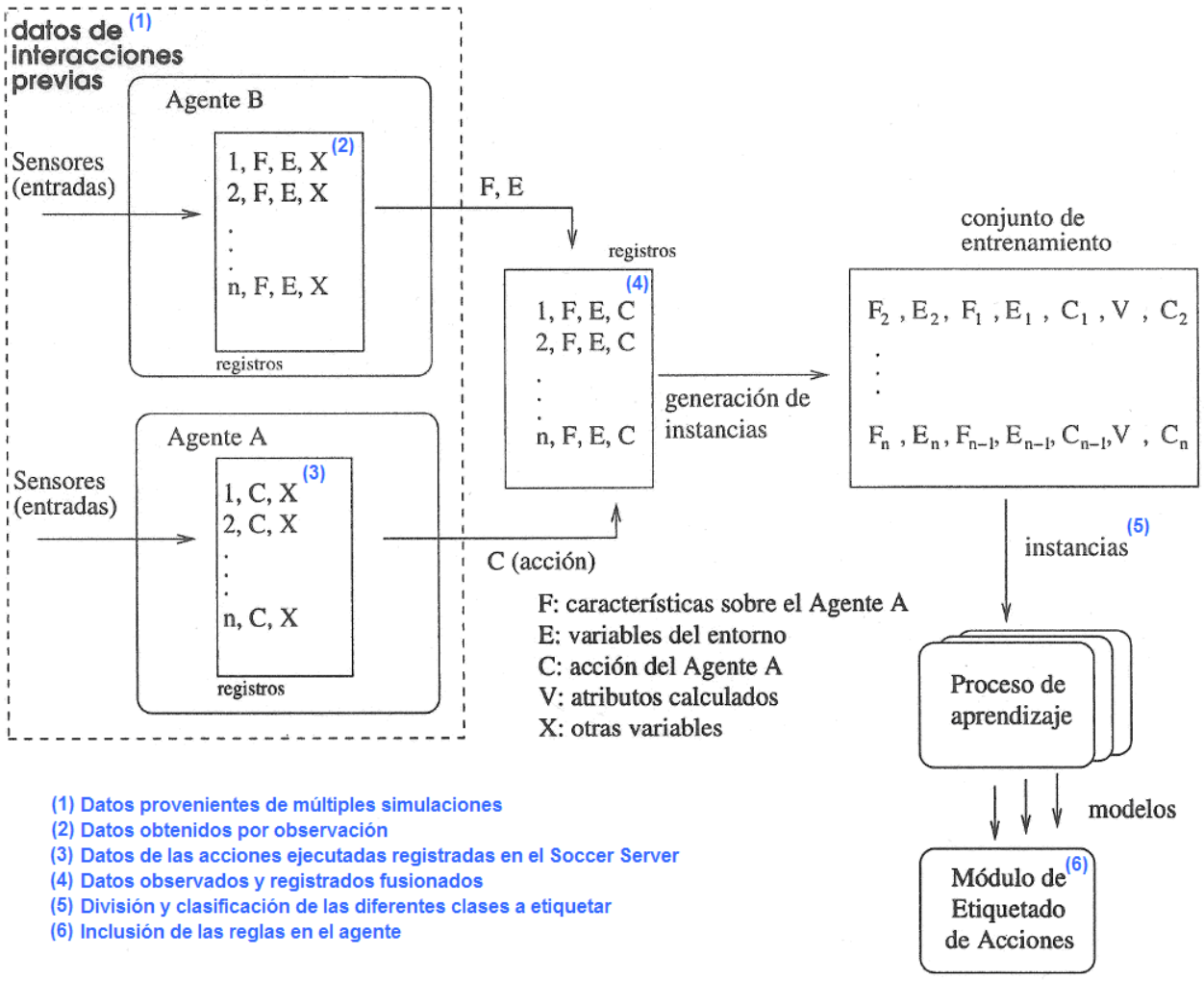


Figura 13: Implementación del Módulo de Etiquetado de Acciones[11]

En el esquema se muestra el resultado del proceso de obtención de datos en el bloque izquierdo, rotulado como “datos de interacciones previas”. El agente B, que es el agente modelador, registra la información percibida del agente rival y del entorno. A su vez el agente A, el agente a modelar, registra a través del *Soccer Server* las acciones que ejecuta. Estos registros se unen las instancias que componen el conjunto de entrenamiento, en el proceso anteriormente definido como fusión de las acciones del agente rival con los datos percibidos. La división por

[11] Implementación del Módulo de Etiquetado de Acciones, imagen basada en Ledezma [3]

componentes de los datos se da en el proceso de aprendizaje, que genera una serie de ficheros de reglas que componen el módulo de etiquetado de acciones o MEA.

Como se puede ver, a la hora de implementar el MEA se han tenido que crear varias utilidades intermedias, pero el esquema general de creación de dicho módulo permanece intacto.

3.2.3 GENERACIÓN DEL MODELADOR

La Figura 14 muestra el proceso de creación del **modelador**, que es relativamente similar al proceso de generación del etiquetador. Se parte del fichero de *log* generado por el delantero, del cual se extraen los datos necesarios para generar tuplas de información ordenada por tiempo. Estos datos son separados por sus distintos componentes, a través de los cuales se obtiene un etiquetador de acciones utilizando la herramienta de minería de datos *Weka*. Al igual que en el proceso de generación del etiquetador, esta herramienta aplica unos algoritmos de clasificación, generando unos ficheros de reglas, que a su vez son adaptadas al lenguaje en el que se ha programado el agente. Una vez convertidas, estas reglas se incorporan al agente para posteriormente ser utilizadas en el proceso de selección del comportamiento de dicho agente durante la ejecución. Más adelante se explicará en profundidad cada uno de los archivos y programas envueltos hasta llegar al resultado final.

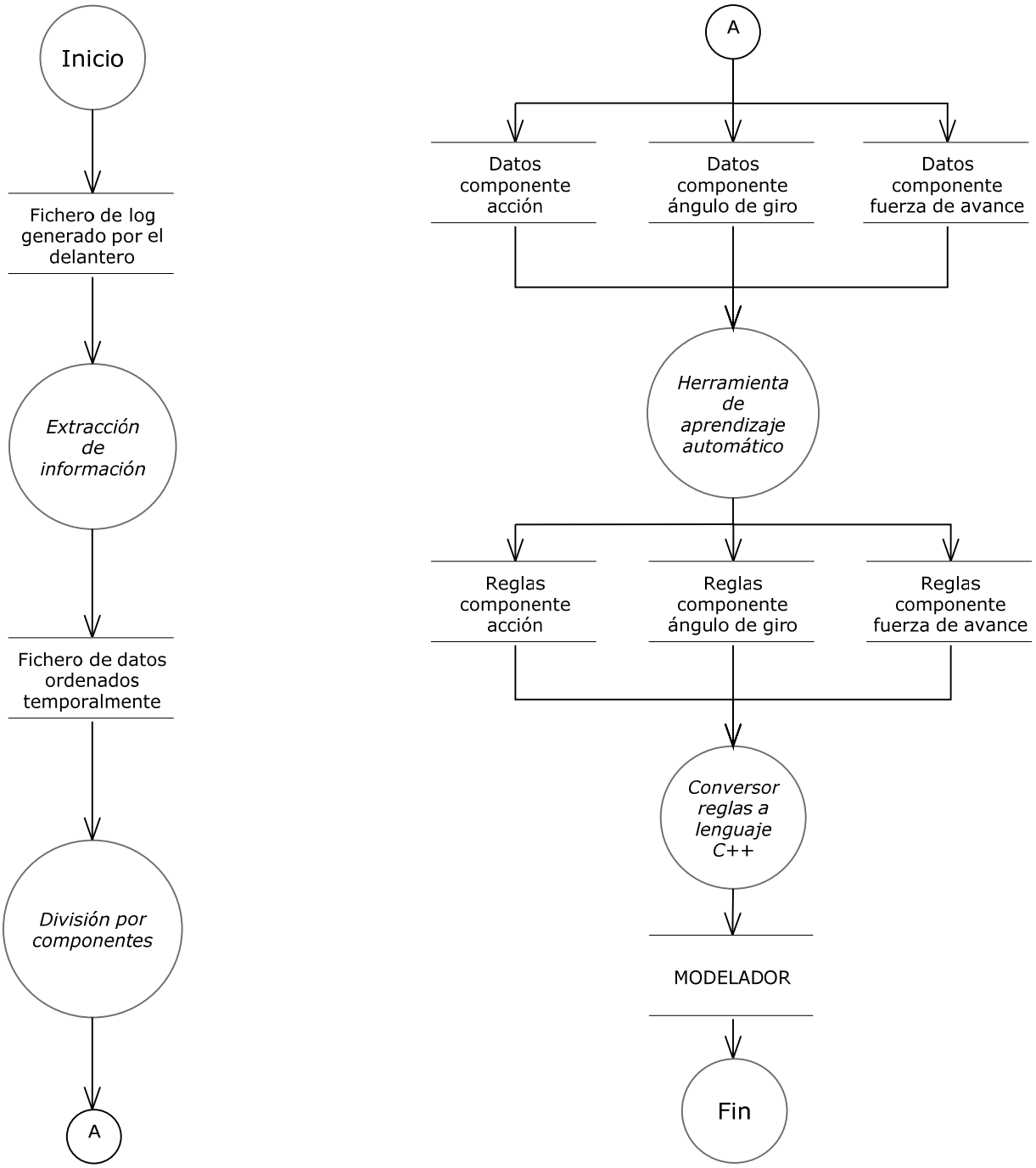


Figura 14: Esquema del proceso de creación del modelador

Ambos procesos tienen muchos elementos en común, puesto que a partir de la división de componentes no hay variación, pero se puede observar que el origen de datos es diferente. Más adelante se mostrará que las herramientas utilizadas en los procesos intermedios son diferentes, puesto que los datos no comparten formato. Además es muy importante considerar que el fichero de *log* de entrada es diferente al del recibido en el etiquetador, puesto que para generar el modelador se utilizan unos parámetros diferentes, como ya se demostró en el apartado 2.7.

3.2.3.1 CREACIÓN DEL MODELADOR: PASO A PASO

El proceso es similar a la construcción del etiquetador, en tanto que parte de datos obtenidos a través de simulaciones, y obtiene un conjunto de reglas que serán incorporadas de nuevo al agente. Pero tanto los datos como el procesamiento son diferentes.

El primer paso para crear el modelador es asegurarse de que el etiquetador está incorporado al agente modelador, puesto que parte de los datos que se utilizan para construir el modelador provienen de información obtenida a través del clasificador, ya que durante la simulación el agente utilizará el conjunto de reglas incorporado en el paso anterior para predecir las acciones que llevará a cabo el agente a modelar (y los parámetros de estas acciones).

En el punto 3.2.2 se muestra un esquema de la construcción del etiquetador. Se considera que el procedimiento de generación del etiquetador y del modelador son en su gran parte análogos, así que este apartado se centrará en explicar los diferentes pasos y las diferencias que existen con el etiquetador, si es que las hubiera. Los puntos a seguir son los siguientes:

- **Obtención de datos:** para crear el modelador también hay que tomar datos de simulación de partidos, así que el agente modelador tendrá que competir de nuevo contra el agente rival. Ahora bien, como los datos del etiquetador ya han sido incorporados al agente, el resultado será diferente, puesto que aprovechará ese clasificador en su beneficio.

  Para asegurarse de que los datos obtenidos son representativos, se procede también a ejecutar 100 ejecuciones de jugadas de ataque por parte del delantero contra el delantero rival. La combinación proporciona un modelo de tamaño análogo al del etiquetador.

  Estas ejecuciones generarán también ficheros de *logs* tanto por parte del propio agente observador como por parte del *Soccer Server*, pero estos últimos no son utilizados, puesto que en este paso no es necesario contar con las acciones realizadas por el agente a modelar. Este cambio se refleja en el fichero de *log* del agente CMUnited 99, que será ligeramente diferente, puesto que a los datos percibidos se le incorporarán también las predicciones de las acciones a llevar a cabo por el agente rival, gracias a la labor del etiquetador. En la Figura 14 este *log* generado es "*Fichero de log generado por el delantero*".

- **Procesamiento de los datos percibidos por el agente CMUnited 99:** en este punto del proceso se trata el fichero de registro obtenido por el propio agente con la información observada y las acciones clasificadas por el etiquetador. Un programa catalogará los datos para obtener trazas de información ordenadas temporalmente, conteniendo cada traza los datos registrados para un instante de tiempo determinado y el instante anterior. Esta forma de combinar la información será de utilidad más adelante a la hora de generar los árboles de reglas de decisión. El proceso correspondiente en la Figura 14 corresponde a

"*Extracción de información*", siendo el fichero resultante "*Fichero de datos ordenados temporalmente*".

- **División por componentes de los datos:** tomando como entrada el fichero resultante del anterior paso, una utilidad parametriza la información. La herramienta correspondiente en la Figura 14 es "*División por componentes*". La información se divide en tres tipos del mismo modo que se hizo en el etiquetador:
    - información global de todas las acciones que realiza el agente a modelar, que corresponde en la Figura 14 al fichero "*Datos componentes acción*".
    - información particular para deducir el ángulo de giro en el caso de que la acción a tomar sea *turn* o girar, que corresponde en la Figura 14 al fichero "*Datos componentes ángulo de giro*".
    - información para deducir la fuerza de avance en el caso de que la acción sea *dash*, que corresponde en la Figura 14 el fichero "*Datos componentes fuerza de avance*".
- **Obtención de reglas de clasificación para cada componente**: la información obtenida en el punto anterior será igualmente tratada por una herramienta de aprendizaje automático, que partiendo de estos datos obtendrá una serie de reglas para deducir los valores de cada una de las clases. El proceso correspondiente en la Figura 14 es "*Herramienta de aprendizaje automático*", y los ficheros resultantes son "*Reglas componente acción*", "*Reglas componente ángulo de giro*" y "*Reglas componente fuerza de avance*".
- **Inclusión de los ficheros de reglas en el agente:** el último paso es transformar esos árboles de reglas al lenguaje C++ para poder integrarlos en el agente. En la Figura 14 esta herramienta corresponde a "*Conversor de reglas a lenguaje C++*".

En el Anexo B: Manual de usuario, se explica en detalle cada uno de estos pasos en profundidad.

Este proceso es referenciable en el esquema de creación del MABO, concretamente con la Figura 9. Si se amplía dicha figura con la información del proceso llevado a cabo para la creación tanto del etiquetador como del modelador se llega a la Figura 15:

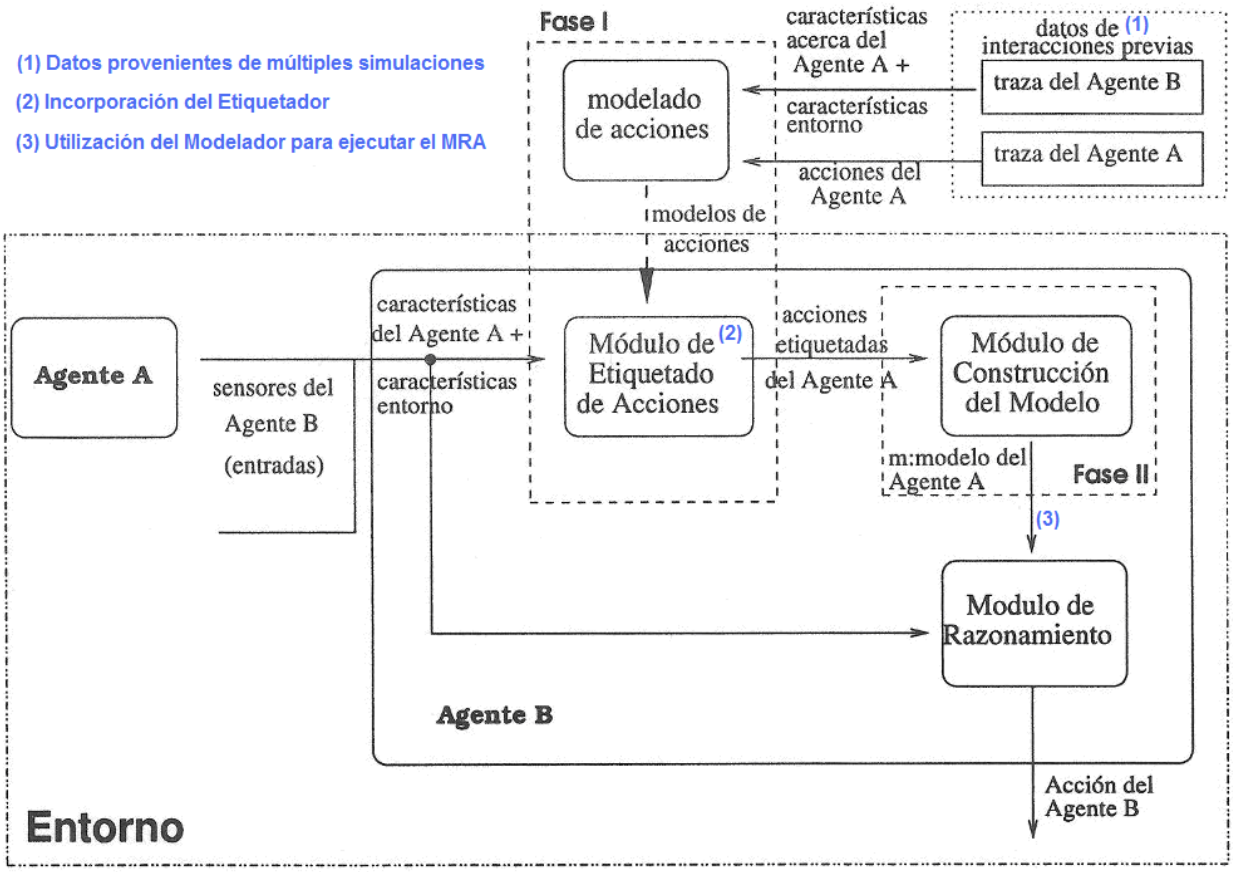


Figura 15: Implementación del Módulo de Construcción del Modelo[12]

La Fase I del esquema, en la parte superior, muestra el proceso de creación del etiquetador, en el que el agente B, que es el agente modelador, obtiene información percibida a través de las simulaciones. Esa información será fusionada junto con el registro de acciones llevadas a cabo por el agente A o agente a modelar, lo que al ser tratada dará como resultado una serie de reglas que clasifican las acciones del agente a modelar. Una vez incorporadas dichas reglas al agente B, se pasa a la Fase II, donde se construye el modelador. El proceso intermedio viene marcado por la utilización de información percibida por el agente B y las acciones etiquetadas por el clasificador incorporado en el punto anterior. El tratamiento de esta información dará lugar a una serie de reglas que componen el modelo del agente A, que a su vez será utilizado por el módulo de razonamiento.

[12] Implementación del Módulo de Construcción del Modelo, imagen basada en Ledezma [3]

3.2.4 CONFIGURACIÓN EXPERIMENTAL

En el presente punto se tratan las actualizaciones que han sido llevadas a cabo buscando la optimización del procedimiento de implementación del MABO.

3.2.4.1 CREACIÓN DE UTILIDADES INTERMEDIAS

Todas las herramientas a las que se hace referencia en este proyecto han sido creadas de cero o actualizadas en mayor o menor medida, siempre con la intención de agilizar y optimizar el proceso de creación del etiquetador/modelador.

3.2.4.2 UTILIZACIÓN DE UN FICHERO DE CONFIGURACIÓN EXTERNO

Con el objetivo de agilizar los procesos de modificación del agente CMUnited 99 y del agente *Trainer*, se ha optado por crear un fichero de configuración externa cuya modificación no requiere de una recompilación del código fuente. La configuración está detallada en el Anexo B.

3.2.4.3 RECARGA DE LA ENERGÍA DEL JUGADOR POR CADA JUGADA REALIZADA

A la hora de realizar N iteraciones con M jugadas cada una, la energía del delantero va disminuyendo poco a poco, con lo que al llevar 4 o 5 jugadas empezaba a cansarse y utilizar movimientos más lentos. La modificación se encuentra detallada en el Anexo C.

3.2.4.4 COLOCACIÓN ALEATORIA DE LOS JUGADORES

Cada nueva jugada el *Trainer* se encarga de colocar a los jugadores en su posición inicial. Para mejorar el aprendizaje y evitar una sobreadaptación del cliente a unas rutinas predefinidas, se ha decidido incluir una función de colocación del delantero con un componente aleatorio. En la siguiente imagen se muestra el área de inicio en el cual puede aparecer el delantero. Otra alternativa plausible consiste en crear un vector de posiciones e ir avanzando entre ellas, o usar una posición del vector al azar. Al igual que el jugador, la pelota también está sujeta a un posicionamiento aleatorio, aunque más guiado que en el posicionamiento del jugador, puesto que se coloca entre el jugador y la portería contraria, aunque no a la misma altura que el jugador. El hecho de que la pelota no se coloque aleatoriamente es una decisión de diseño con vistas a la optimización, por las siguientes razones:

- La posición del jugador pasa a ser irrelevante: tiene que moverse hasta llegar a la pelota, con lo que su posición inicial podría ser fija.
- Se alargan las ejecuciones al tener que sumar el tiempo que requiere el agente CMUnited 99 en recorrer el campo para acceder a la pelota. El tiempo necesario es de tres segundos en el peor de los casos, por lo que en cien ejecuciones con treinta iteraciones cada una tendría un impacto total de dos horas y media a la hora de realizar las pruebas.
- El comportamiento de avanzar con la pelota cuando se está fuera de la distancia de tiro obliga al jugador a chutar la bola en dirección a portería (para avanzar rápidamente), y si el

posicionamiento del jugador (que es subjetivo en base a los marcadores del entorno) no es lo suficientemente preciso el chut que partía con la intención de acercarse a la portería puede acabar con la bola en los dominios del portero, invalidando la jugada. Es decir, si el balón está lo suficientemente cerca de la portería, es muy probable que el primer golpeo del agente CMUnited 99 al balón lo envíe al área de influencia del portero, perdiéndolo en ese caso.

En la Figura 16 se observa el campo con el rango de posiciones posibles del jugador, pintadas sobre fondo negro en la parte izquierda de la imagen, con una posición ideal resaltada con los colores originales, y el rango de ubicaciones del balón respecto al jugador, delimitados por la elipse azul sobre fondo negro, con una posición centrada resaltada.

Figura 16: Colocación aleatoria del jugador y de la pelota

3.2.4.5 CARGA DE DIFERENTES TIPOS DE FICHEROS DE CONFIGURACIÓN DEL SERVIDOR

La plataforma RoboCup *Soccer Server* permanece en constante evolución, y aunque cada versión mantiene retrocompatibilidad con las anteriores, hay ciertos cambios que han de ser tenidos en cuenta. Uno de ellos es la diferencia de parámetros del archivo de configuración, añadiéndose nuevos elementos con el paso de las versiones. Desafortunadamente el agente CMUnited 99 está diseñado para leer un formato específico de dicho archivo de configuración, por lo que se ha optado por dar la posibilidad de leer el archivo de configuración de las últimas versiones del *Soccer Server*. Este proceso está explicado en profundidad en el Anexo B.

3.2.4.6 MODIFICACIÓN DEL COMPORTAMIENTO DEL CLIENTE CMUNITED 99

Debido a una estrategia de diseño, el comportamiento de avanzar con el balón controlado del agente delantero consistía en un lento avance a base de pequeñas patadas al balón, con lo que se ganaba en una mejor elección del momento de tiro, ya que la pelota estaba siempre al alcance del delantero, pero se perdía en previsibilidad, debido a que al tener más tiempo de preparación el portero podía colocarse mejor ante el delantero, cubriendo más eficazmente la portería. Además esta lentitud ocasionaba un 33% más de tiempo a la hora de realizar las pruebas.

Se ha optado por una modificación del movimiento, optimizando la conducción del balón mientras el delantero esté lo suficientemente lejos de la portería como para que una patada al balón no lo mande al alcance del portero. Esta optimización se basa en una predicción del área de alcance del portero, con la intención de no enviar el balón a una zona donde seguramente se va a perder en beneficio de aquél.

Para no cambiar el comportamiento general del delantero se ha optado por retomar el enfoque tradicional, disminuyendo la velocidad del agente delantero al acercarse al área rival, con el objetivo de no pasarse de largo, y de posibilitar el tiro, puesto que únicamente se puede producir esa acción cuando el balón está muy cerca del agente.

Así mismo se ha decidido guiar el proceso para forzar el tiro cuando la distancia al centro de la portería es lo suficientemente cercana como para que seguir avanzando sea un riesgo. La solución final muestra un avance ágil del delantero, evitando los problemas descritos anteriormente y consiguiendo más goles, por lo que la sensación general es que en un entorno con defensas éste delantero tendría opciones reales de marcar. Se puede encontrar más información acerca de las modificaciones realizadas en el Anexo C.

3.2.4.7 UTILIZACIÓN DE CLASES NUMÉRICAS

A la hora de generar datos para obtener tanto el etiquetador como el modelador, el fichero de *log* generado por el delantero con los datos del portero tiene un formato específico con múltiples atributos, entre ellos la clase escogida por el delantero (*none*, *dash*, *turn*, *kick*, etc). Esta clase ha sido cambiada a formato numérico para facilitar la inclusión del etiquetador/modelador generado a través de los clasificadores de los que dispone *Weka* en el propio cliente. Esta modificación afecta a varios de los ficheros descritos en el Anexo B, y para llevarla a cabo se han elaborado versiones específicas de algunos programas para utilizar estas clases de tipo numérico. Así pues, se dispone de los programas *ticker_numerico*, *ticker_predict_numerico*, *makeArff_numerico* y *makeArff_predict_numerico*, que son las evoluciones de los programas *ticker*, *ticker_predict*, *makeArff* y *makeArff_predict*, respectivamente.

3.2.4.8 REFINAMIENTO DE LOS FICHEROS DE REGLAS

Los clasificadores obtenidos a través del programa de minería de datos *Weka* están compuestos con una sintaxis específica que hace que existan ciertas incompatibilidades a la hora de transferir las reglas obtenidas al agente CMUnited 99. Es decir, para incorporar el etiquetador o el modelador al agente hay que utilizar una herramienta específicamente desarrollada para este proceso. Más información de esta herramienta se puede encontrar en el Anexo B. Este programa, siguiendo los pasos que definen el MABO, analiza los conjuntos de reglas que definen el etiquetador y el modelador justo en el momento anterior a incorporarlas en el agente, y elimina los errores que hayan podido producirse.

4 RESULTADOS EXPERIMENTALES

Como se comentó en la sección 1.2, el objetivo del proyecto es doble, por un lado la optimización del proceso de modelado del MABO y por otra parte la verificación de la creación de modelos. En este punto se procederá a identificar y valorar los resultados obtenidos en ambos procesos.

4.1 OPTIMIZACIÓN DEL PROCESO DE MODELADO

Una vez llevada a cabo la implementación del MABO en el agente CMUnited 99, se ha de estudiar si los resultados obtenidos son equivalentes a aquellos de los que se disponía al principio del proyecto, es decir, comprobar que las nuevas utilidades proporcionan los resultados esperados.

Debido a que este proyecto se basa en Ledezma [3], se pusieron a disposición de este alumno varios conjuntos de pruebas con sus respectivos resultados, a la vez que el proceso completo inicial de implementación del modelo. Así mismo el equipo rival que se utilizó en aquella ocasión también fue utilizado durante la verificación de resultados (ver apartado 4.2.3), con lo que una comprobación completa pudo hacerse efectiva.

Así pues, comentar que una vez incorporadas las distintas utilidades y las optimizaciones de código realizadas, tanto la generación del etiquetador como la del modelador se llevaron a cabo en los términos esperados.

4.1.1 VERIFICACIONES REALIZADAS

Las pruebas llevadas a cabo para comprobar la optimización del proceso de modelado son las siguientes:

4.1.1.1 VALIDACIÓN DEL FICHERO DE ACCIONES REALIZADAS POR EL AGENTE A MODELAR

Como se ha comentado en el punto 3.2.1.2 Obtención de la información del agente rival, la información relativa a las acciones del agente a modelar es obtenida a través del fichero de *log* emitido por el *Soccer Server*. La información obtenida a través de este fichero requiere un proceso de filtrado intermedio que ha sido automatizado, pero ha sido necesario verificar que los resultados obtenidos son los esperados.

Gracias a los ficheros originales que componían el desarrollo del MABO [3] se ha podido garantizar que los resultados obtenidos han sido los esperados. Todos los ficheros de pruebas proporcionados generaron los resultados esperados.

4.1.1.2 OBTENCIÓN CORRECTA DE LOS ATRIBUTOS CALCULADOS

Los atributos calculados son aquellos atributos obtenidos a través de operaciones aritméticas sobre los atributos percibidos por el agente observador. En el proceso de creación del etiquetador 3.2.2 se explica que hay un paso para fusionar los datos percibidos por el agente modelador con

los datos de las acciones realizadas por el agente a modelar. Para ayudar en el proceso del clasificador, a dichos atributos se les añade una serie de atributos calculados, como pueden ser variaciones en el tiempo de atributos respecto al instante anterior.

En el desarrollo inicial del MABO esta generación de atributos calculados consistía en una hoja de cálculo que debía ser rellenada manualmente. Este proceso ha sido automatizado, y se ha podido confirmar que la generación ha sido realizada correctamente con los registros iniciales de los que se dispone.

4.1.1.3 GENERACIÓN DE FICHEROS VÁLIDOS PARA LA HERRAMIENTA DE APRENDIZAJE AUTOMÁTICO *WEKA*

Una de las necesidades planteadas fue el desarrollo de una utilidad para generar automáticamente ficheros legibles por dicha herramienta de minería de datos. Además dichos ficheros debían contar con una separación, generando árboles de decisión según la acción a tomar por el rival, proporcionando la fuerza de avance en caso de que fuera esa la acción tomada, o proporcionando el ángulo de giro en caso de que la respuesta del agente rival fuera la de girar. Al contar con los datos de origen proporcionados por la herramienta anterior, la forma de comprobar que esta utilidad daba los resultados adecuados fue comprobar que el conjunto de reglas generadas por la herramienta de minería de datos *Weka* era iguales en ambos casos, situación que efectivamente tuvo lugar.

4.1.1.4 RELACIÓN CORRECTA ENTRE LOS PORCENTAJES DE ACIERTO A LA HORA DE CLASIFICAR

Un dato importante es el porcentaje de éxito al clasificar obtenido utilizando los algoritmos de clasificación de la herramienta de minería de datos *Weka*. Realizando simulaciones completas desde el principio contra el mismo agente rival descrito en el punto 4.2.3, se comprobó que efectivamente los resultados a la hora de generar el etiquetador o el modelador eran parecidos. Este procedimiento se realiza comparando el porcentaje de acierto al clasificar el conjunto de datos de acciones del agente a modelar antes y después de aplicar todas las modificaciones al cliente, tanto en la generación del etiquetador como en la creación del modelo. Del mismo modo se comparan los índices de correlación de los conjuntos de datos de la clase *Turn* o ángulo de giro y de la clase *Dash* o fuerza de avance. Al comprobar dichos índices con los obtenidos en las simulaciones iniciales se pudo comprobar que eran de órdenes similares, debido a la aleatoriedad de los orígenes de datos, pero en todo caso muy parecidos.

4.1.1.5 COMPORTAMIENTO IDÉNTICO DEL AGENTE CMUNITED 99

Debido a la modificación del agente modelador para incluir las configuraciones experimentales definidas en el punto 3.2.4, una de las pruebas a realizar consiste en comprobar que el comportamiento del agente CMUnited 99 de partida y el obtenido al realizar las modificaciones sea idéntico. A la hora de simular ejecuciones de partidos, se puede comprobar que el agente

CMUnited 99 original es sustancialmente más lento y menos efectivo que el agente modificado, debido a que los algoritmos de movimiento y de selección de tiro han sido modificados ligeramente. Estos cambios en el comportamiento son debidos a las optimizaciones en el código del agente llevadas a cabo para agilizar el proceso, por lo que esta prueba no es del todo representativa. Se puede obtener más información de esta modificación en el Anexo C.

Un aspecto que ha sido mantenido es el de la estabilidad. Se ha puesto especial énfasis en que las modificaciones en el agente no conlleven una tasa de errores asociadas, por lo que todos los cambios han sido probados en cientos de simulaciones para asegurar este punto.

4.1.1.6 AGILIZACIÓN EN EL TIEMPO EMPLEADO EN LA SIMULACIONES

A costa de incumplir el punto anterior, se ha puesto un especial énfasis en agilizar el proceso de simulación de ejecuciones. Esta prueba se realiza lanzando ejecuciones del agente CMUnited 99 antes y después de la evolución contra el mismo rival, y comparando el tiempo obtenido. Las modificaciones del nuevo agente hacen que el tiempo necesario sea significativamente menor, empleando aproximadamente la mitad de tiempo. Las modificaciones experimentales que han llevado a esta agilización del proceso son descritas en los puntos 3.2.4.4 y 3.2.4.6. La ganancia obtenida en tiempo es de algo menos de 10 segundos por minuto para el equipo de desarrollo expuesto en el Anexo B, dándose un incremento de la rapidez de aproximadamente un 20%. Información exacta de tiempos se puede obtener en el punto 4.2.2.2.

4.1.1.7 TIEMPO TOTAL EMPLEADO EN LA CREACIÓN E INCORPORACIÓN DEL ETIQUETADOR Y DEL MODELADOR

Realizando una medición del tiempo total empleado en llevar a cabo todos los procesos del etiquetador/modelador, se demuestra sin duda que la incorporación de las nuevas utilidades al proceso de implementación del MABO consigue obtener unos tiempos sustancialmente inferiores. La utilización de las herramientas descritas en el punto 4.1.1.1, 4.1.1.2, y 4.1.1.3, y la configuración experimental realizada en el punto 3.2.4.8 han conseguido agilizar y facilitar estos procesos. No es posible dar una medida aproximada de la ganancia en tiempo, puesto que muchos de los procesos se hacían a mano, tomando los datos resultantes de una herramienta, copiándolos en una hoja de cálculo y aplicando fórmulas sobre ellos para obtener los datos necesarios como entrada de otra herramienta.

4.1.1.8 RESULTADOS CUANTIFICABLES DE LA SIMULACIÓN

Esta comprobación se lleva a cabo comparando los resultados de una simulación promedio, con el agente CMUnited 99 original compitiendo contra un determinado agente, con otra simulación promedio del agente CMUnited 99 modificado contra el mismo agente, comprobando estos datos con y sin modelo incorporado. Debido a las modificaciones realizadas, la cantidad de goles encajados por el portero rival es ligeramente inferior compitiendo contra el agente CMUnited 99

original que contra el agente modificado. Esto se puede comprobar con los datos vertidos en el punto 4.2.3.3, mientras que el agente original marca de media unos 3 goles, el agente modificado se queda en 1'55 goles. Este comportamiento es asumible debido a que tanto con el agente original como con el agente modificado al añadir el modelo hay una mejora sustancial de la cantidad de goles marcados, con lo que no afecta al objetivo del proyecto.

4.1.2 RESULTADO FINAL DE LA OPTIMIZACIÓN DEL MODELO

En base a los puntos anteriores, se puede afirmar que la implementación del modelo ha cumplido con los requisitos, puesto que las diferentes utilidades creadas realizan efectivamente la función para la que se diseñaron, y además lo hacen en menos tiempo, con lo que la implementación del MABO en el agente modelador se facilita enormemente.

4.2 ADQUISICIÓN Y VERIFICACIÓN DE MODELOS

Esta segunda parte de la memoria trata de, una vez confirmado que el proceso de creación del modelo es correcto, comprobar los resultados de la utilización del MABO compitiendo contra agentes rivales, observando de esta manera las mejoras en rendimiento del agente una vez incorporado el modelo.

4.2.1 CLIENTES UTILIZADOS

Se han realizado pruebas con seis tipos diferentes de agentes portero para asegurarse de que no existía una relación de adaptación a un único portero. Para ello se han tomado los porteros de los siguientes equipos:

1. **UvA Trilearn 2003**: equipo ganador de la competición realizada en 2003 en Pádova, desarrollado por la Universidad de Carnegie Mellon, Pittsburgh, Pensilvania, Estados Unidos. Es una evolución del cliente CMUnited 99, aunque no está basado directamente en este agente [38].
2. **YowAI 2004**: equipo que quedó en la novena posición de la competición realizada en 2004 en Lisboa, desarrollado por el laboratorio Takeuchi, de la Universidad de Tokio, Japón [39].
3. **Virtual Werder 2004**: equipo que no consiguió clasificarse para la fase final de la competición del 2004, desarrollado por la Universidad de Bremen, Alemania [40].
4. **Wright Eagle 2005**: equipo desarrollado por la Universidad de Ciencias y Tecnología de China [41], que participó en la competición del año 2005 terminando en segunda posición.
5. **Tokio Tech SFC 2005**: equipo que terminó en la tercera posición de la competición del 2005 realizada en Osaka, Japón. El equipo fue desarrollado en el Instituto Tecnológico de Tokio, en Japón [42].
6. **FC Portugal 2011**: equipo que participó en la competición del año 2011 en Estambul, Turquía, con un séptimo puesto final. Desarrollado conjuntamente entre el Laboratorio de Inteligencia Artificial y Ciencias de la Computación de la Universidad de Oporto [43], y el Instituto de Ingeniería Electrónica y Telemática de la Universidad de Aveiro [44], ambos de Portugal.

La explicación de la relativa antigüedad de la mayoría de los agentes utilizados remite a una mera cuestión práctica. La tesis en la que se basa este proyecto se realizó en 2004, y el portero contra el que se realizaron en su momento las pruebas fue el UvA Trilearn del año 2003. Así pues, se decidió tomar este agente para tener como referencia los resultados que en su momento fueron obtenidos, a la vez que para comprobar que las herramientas modificadas y las nuevas herramientas seguían conservando la intencionalidad original.

La decisión que motivó a utilizar los agentes YowAI y Virtual Werder radica en que tienen el código fuente liberado, con lo que sería posible la realización del algún cambio en el supuesto caso de que hubiera sido necesario. En las reglas de la RoboCup no existe ninguna cláusula que obligue a los equipos participantes a liberar el código fuente de sus proyectos, por lo que no todos los equipos siguen esta práctica. Además este comportamiento hermético se ha acentuado estos últimos años puesto que el nivel de la competición está aumentando considerablemente. Así pues se decidió por tomar dos equipos del año 2004 que están perfectamente documentados. Un detalle a tener en cuenta es que esta diferencia de antigüedad no impide que el sistema sea totalmente compatible con equipos actuales.

Para el resto de los agentes utilizados se ha tomado un enfoque diferente. Los agentes Tokyo Tech y Wright Eagle se han escogido porque quedaron en posiciones muy destacadas de la competición del año 2005, consiguiendo una tercera y una segunda posición respectivamente. En ambos casos sus desarrolladores no liberaron el código fuente, aunque sí sus archivos binarios.

Además se ha escogido un equipo del año 2011, el agente FC Portugal, para comprobar la validez del sistema de aprendizaje contra un equipo actual. Al igual que en los equipos del año 2005, únicamente los archivos binarios están disponibles para descarga. Este equipo ha sido seleccionado por razones de estabilidad a la hora de la experimentación.

En los resultados se muestra que cada portero tiene unas capacidades diferentes, partiendo de los buenos resultados que ofrece el agente UvA Trilearn, con apenas un 5% de goles encajados, hasta llegar al agente Virtual Werder, que llega al 66% de goles recibidos.

Esta selección de porteros confirma que cuanto más restrictivo es el entorno, es decir, mayor dificultad a la hora de anotar, mayor es la mejora obtenida al incorporar el modelo en el agente, lo cual además es de esperar puesto que contra un portero que encaja casi la totalidad de los disparos hay muy poco margen de mejora.

4.2.2 CONSIDERACIONES INICIALES

En este punto se exponen las condiciones iniciales que se llevaron a cabo para realizar la experimentación, así como la información necesaria para entender los resultados obtenidos durante la misma.

4.2.2.1 CONDICIONES DE SIMULACIÓN

Como se ha explicado en el punto 3.2.1,la obtención de datos se hace a través de la simulación de partidos entre los agentes participantes. La forma de realizarlo es explicada en profundidad en el Anexo B. Básicamente consiste en la utilización de unos scripts que ejecutan el servidor *Soccer Server* junto con los agentes y los pone a competir entre ellos, almacenando los resultados de las ejecuciones.

Así pues, para comprobar los resultados de la creación del etiquetador y del modelador se utilizan dichos scripts, que simulan rondas de partidos, permitiendo comprobar de una forma directa la diferencia de incluir o no el modelo obtenido en el agente delantero, que es la medida más precisa de verificar la adaptación del agente al entorno. Para ello se comparan los goles metidos por el agente, o los tiros que han ido fuera, medidas ambas de la precisión que alcanza el agente.

A la hora de realizar las pruebas se han tomado las siguientes consideraciones:

- Para la creación del etiquetador se ha tomado datos de dos iteraciones completamente separadas de 50 jugadas cada una, es decir, de 100 partidos entre el delantero y el portero.
- Estos datos generan un conjunto de alrededor de 4500 instancias para clasificar la acción, y alrededor de 2000 instancias tanto para determinar el ángulo de giro como la fuerza de avance. Se estima que esta cantidad de datos es suficiente para generar el etiquetador y el modelador.
- Estas instancias darán lugar al etiquetador, que será añadido al agente. Posteriormente se repetirán los dos primeros puntos pero esta vez con la configuración necesaria para elaborar el modelador. Una vez creado e incorporado al agente, se realizan las pruebas para comprobar la habilidad del agente delantero.
- A la hora de hacer las pruebas para comparar la habilidad del agente con y sin modelo se intenta evitar en la medida de lo posible la alta aleatoriedad del entorno, por lo que se ha decidido optar por ejecuciones de 100 iteraciones cada una, cada una de ellas de 30 jugadas, es decir, 3000 iteraciones. Al hacer tantas iteraciones se elimina el problema de la aleatoriedad en el entorno, que viene definida por el hecho de que el delantero y el balón se colocan aleatoriamente en el campo, además de que el servidor introduce ruido en las señales para simular condiciones lo más próximas a la realidad.

Así pues para la verificación de resultados se han de hacer los siguientes pasos:

1. Realizar 2 iteraciones de 50 jugadas con el agente modelador contra el portero rival.
2. Generar el etiquetador con los datos obtenidos del punto 1. Incorporar el etiquetador al agente delantero.
3. Realizar 2 iteraciones de 50 jugadas con el agente obtenido por el punto 1 contra el portero rival. La configuración del agente habrá cambiado, por lo que los datos obtenidos serán diferentes.
4. Generar el modelo de comportamiento del portero con los datos obtenidos en el punto 3. Incorporar el modelo al agente delantero.
5. Realizar 100 iteraciones de 30 jugadas con el agente obtenido en el punto 5.

6. Realizar 100 iteraciones de 30 jugadas con el agente modelador, en su estado inicial, contra el portero rival.
7. Comparar los resultados de goles a favor y paradas realizadas por el portero rival en los puntos 5 y 6.

4.2.2.2 TIEMPO EMPLEADO EN LA SIMULACIÓN

A la hora de llevar a cabo los pasos 5 y 6 no se encuentran grandes diferencias de tiempos. En relación al punto 4.1.1.6, con las modificaciones aplicadas al agente modelador, una simulación en la que los agentes son colocados en el terreno de juego 30 veces, es decir, 30 oportunidades de ataque, tiene una duración media de alrededor de 3 minutos y medio, por lo que en un lote de 100 simulaciones puede durar casi 6 horas. En el caso de emplear el agente modelador sin las mejoras expuestas, la misma simulación de 30 jugadas tiene una media de duración de algo menos de 4 minutos y 10 segundos, por lo que la simulación completa dura casi 7 horas, que es casi un 20% más de tiempo. Estas mediciones se han llevado a cabo con el equipo de desarrollo explicado en el Anexo B.

En la Tabla 3 se muestran los resultados de tiempo en simulaciones llevadas a cabo contra el mismo portero.

DELANTERO UTILIZADO	TIEMPO EN EJECUCIÓN	
	30 JUGADAS	30000 JUGADAS
CMUnited 99 [3]	248'4 segundos	6'9 horas
CMUnited 99 con las mejoras	207'47 segundos	5'78 horas

Tabla 3: Tiempo empleado en la simulación

4.2.2.3 INFORMACIÓN OBTENIDA EN LA SIMULACIÓN

En el punto 4.2.3 y posteriores se muestran los resultados obtenidos de realizar el proceso anterior con distintos agentes rivales. Estos resultados se muestran en unas tablas que contienen los diferentes parámetros observados. La información mostrada es la siguiente:

1. *Action*: muestra el porcentaje de acierto del clasificador cuando el parámetro que se desea obtener es la acción llevada a cabo por el agente a modelar. A mayor porcentaje mejor clasificación.
2. *Turn angle*: muestra el coeficiente de correlación del clasificador cuando el parámetro que se desea obtener es el ángulo de giro que muestra el agente a modelar cuando su acción es *turn* o girar.
3. *Dash power*: muestra el coeficiente de correlación del clasificador cuando el parámetro que se desea obtener es la fuerza con la que avanza el agente a modelar cuando su acción es *dash* o avanzar.

Es necesario considerar que el coeficiente de correlación es un índice estadístico que muestra la relación lineal entre dos variables cuantitativas (en este caso el conjunto utilizado como muestra y las funciones obtenidas por el clasificador) que toma valores entre -1 y 1, siendo 1 una correlación lineal positiva perfecta (si una variable aumenta la otra lo hace en una proporción idéntica), 0 que no existe relación lineal entre variables y -1 que tienen una relación inversa entre ellas, con lo que si el coeficiente de correlación es menor que 0, el clasificador generado no tendrá ninguna utilidad.

Por último, para entender los resultados de la simulación hay que considerar las siguientes formas de acabar la jugada:

- *Gol*: el delantero chuta el balón atravesando la línea que marca la portería.
- *Parada del portero*: el portero se hace con la pelota, bien porque el avance del delantero con la bola es incontrolado y entra la pelota en el área debido a un auto pase y llega antes el portero, o bien porque el delantero tira a puerta y el portero para el disparo.
- *Tiros fuera*: tiros que han salido por la línea de fondo debido a que el cálculo del delantero ha sido erróneo y la pelota no ha alcanzado portería.
- *Fallos de simulación*: ninguno de los casos anteriores. Se da cuando el tiempo de la jugada excede del tiempo máximo impuesto por el *Trainer*. Este caso se da principalmente cuando el delantero chuta con fuerza, el balón golpea en uno de los palos de la portería y sale disparado hacia atrás. En la mayoría de estos casos el balón sale disparado a la mitad de la cancha, con lo que el tiempo que tarda en llegar el delantero a la bola, volver a orientarse y avanzar hacia la portería a veces supera el tiempo límite establecido para cada simulación, por lo que se dan estos casos

A continuación se describen los experimentos realizados con los diferentes agentes rivales. La manera de ejecutar un portero u otro se detalla en Anexo B: Planificación y presupuesto.

4.2.3 MODELADO DEL AGENTE UVA TRILEARN

En principio el agente UVA Trilearn 2003 es el agente portero con mejor tasa de acierto, lo que implica una menor tasa de goles encajados. Así pues, las posibilidades de mejora son relativamente amplias.

Los siguientes puntos muestran los resultados obtenidos en la creación del etiquetador, en la generación del modelador, y los resultados de la simulación, además de una comparativa de la ganancia obtenida.

4.2.3.1 CREACIÓN DEL ETIQUETADOR

En la Tabla 4 se muestran los resultados de la creación del **etiquetador** en el agente UvA Trilearn.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	77,29%
Turn angle	0,206
Dash power	0,597

Tabla 4: Resultados de la creación del etiquetador para el agente UvA Trilearn

Como se puede comprobar, el porcentaje de acierto en la clasificación del parámetro acción es cercano a un 80%, mientras que los índices de correlación han obtenido resultados más discretos, con el ángulo de giro en un 0,206 que indica una correlación baja.

4.2.3.2 CREACIÓN DEL MODELADOR

En la Tabla 5 se pueden observar los resultados de la creación del **modelador** en el agente UvA Trilearn.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	78,97%
Turn angle	0,622
Dash power	0,837

Tabla 5: Resultados de la creación del modelador para el agente UvA Trilearn

A la hora de crear el modelador todos los parámetros han sufrido un incremento en sus índices, siendo muy relevante este incremento en los índices de correlación de *Turn* y *Dash*, que han quedado muy cercanos a 1. Esto indica que el clasificador generado se corresponde muy bien con los datos proporcionados, con lo que es de esperar una mejora a la hora de la simulación.

4.2.3.3 RESULTADOS DE LA SIMULACIÓN

La Tabla 6 y la Tabla 7 muestran información de los resultados obtenidos al realizar las simulaciones con y sin modelo del agente UvA Trilearn, empezando por la **simulación sin modelo**, es decir, la media de las jugadas realizadas entre el delantero por defecto y el portero Uva Trilearn.

PARÁMETRO	VALOR OBTENIDO
Goles	1,55
Paradas portero	25,51
Tiros fuera	2,89
Fallos de simulación	1 fallo

Tabla 6: Resultados de la simulación sin modelo del agente UvA Trilearn

A continuación la Tabla 7 muestra los resultados de la **simulación al añadir el modelo generado al agente CMUnited 99**, es decir, la media de las jugadas realizadas entre el delantero al que se le ha añadido el modelo del rival y el portero Uva Trilearn.

PARÁMETRO	VALOR OBTENIDO
Goles	4,97
Paradas portero	21
Tiros fuera	3,97
Fallos de simulación	1 fallo

Tabla 7: Resultados de la simulación con modelo del agente UvA Trilearn

Como se ve, ha habido una mejoría muy importante en la relación de goles anotados por el agente con el modelo incorporado. Objetivamente se muestra que un porcentaje de tiros que antes detenía el portero ahora acaban siendo gol.

4.2.3.4 GANANCIA EN LA SIMULACIÓN POR LA INCLUSIÓN DEL MODELO

La Tabla 8 muestra la ganancia observada en la simulación.

PARÁMETRO	GANANCIA NETA	GANANCIA PORCENTUAL
Goles	+3,42 goles	+220,65%
Paradas del portero	-4,51 paradas	-17,68%

Tabla 8: Ganancia en la simulación debida a la inclusión del modelo en el agente UvA Trilearn

En vista de los resultados, se puede determinar que, al incorporar el modelo del portero rival, el delantero marca más del doble de goles, con lo que la ganancia en la simulación es más que evidente.

4.2.3.5 DIFERENCIA RESPECTO AL AGENTE CMUNITED 99 ORIGINAL

En este apartado se muestran de manera informativa los valores obtenidos con el agente CMUnited 99 original llevados a cabo en los mismos procesos en Ledezma [3], comparándolos con el agente CMUnited con las modificaciones realizadas en el presente proyecto, vistas en el punto 3.2.4.

La Tabla 9 muestra los resultados de la creación del etiquetador. El valor del clasificador de la acción es exactamente el mismo, mientras que los índices de correlación varían.

TIPO DE CLASIFICADOR	VALOR OBTENIDO AGENTE ORIGINAL	VALOR OBTENIDO AGENTE MODIFICADO
Action	77,29%	77.14%
Turn angle	0,206	0.32
Dash power	0,597	-0.04

Tabla 9: Resultados de la creación del etiquetador para el agente UvA Trilearn original - modificado

La Tabla 10 muestra los resultados de la creación del modelador. Los valores obtenidos son próximos a los originales, pero no guardan una relación directa.

TIPO DE CLASIFICADOR	VALOR OBTENIDO AGENTE ORIGINAL	VALOR OBTENIDO AGENTE MODIFICADO
Action	78,97%	85.4%
Turn angle	0,622	0.45
Dash power	0,837	0.76

Tabla 10: Resultados de la creación del modelador para el agente UvA Trilearn original - modificado

La Tabla 11 y la Tabla 12 muestran información de los resultados obtenidos al realizar las simulaciones con y sin modelo del agente UvA Trilearn, empezando por la **simulación sin modelo**, es decir, la media de las jugadas realizadas entre el delantero por defecto y el portero Uva Trilearn.

PARÁMETRO	VALOR OBTENIDO AGENTE ORIGINAL	VALOR OBTENIDO AGENTE MODIFICADO
Goles	1,55	0.38
Tiros fuera	2,89	4.64

Tabla 11: Resultados de la simulación sin modelo del agente UvA Trilearn original - modificado

A continuación la Tabla 12 muestra los resultados de la **simulación al añadir el modelo generado al agente CMUnited 99**, es decir, la media de las jugadas realizadas entre el delantero al que se le ha añadido el modelo del rival y el portero Uva Trilearn.

PARÁMETRO	VALOR OBTENIDO AGENTE ORIGINAL	VALOR OBTENIDO AGENTE MODIFICADO
Goles	4.97	2.34
Tiros fuera	3.97	5.85

Tabla 12: Resultados de la simulación con modelo del agente UvA Trilearn original - modificado

Las dos últimas tablas presentan menos campos de los obtenidos en el punto 4.2.3.3, pero presentan suficiente información para asegurar que los valores, obtenidos antes y después de modificar el agente, son ligeramente diferentes. La variación de goles encajados es significativa en términos cuantitativos, pero en esa mínima cantidad de goles encajados se comprueba que el comportamiento del agente no ha variado demasiado, porque comparado con el resto de agentes sigue siendo el que recibe menos goles.

4.2.4 MODELADO DEL AGENTE YOWAI

El siguiente portero objeto de estudio es el agente YowAI 2004. Este agente se caracteriza por tener una tasa de goles encajados media, con lo que la mejora en principio no será tan evidente como en el agente UVA Trilearn.

Los siguientes puntos muestran los resultados obtenidos en la creación del etiquetador, en la generación del modelador, y los resultados de la simulación, además de una comparativa de la ganancia obtenida.

4.2.4.1 CREACIÓN DEL ETIQUETADOR

En la Tabla 13 se muestran los resultados de la creación del **etiquetador** en el agente YowAI.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	68,54%
Turn angle	0,202
Dash power	0,561

Tabla 13: Resultados de la creación del etiquetador para el agente YowAI

Como se puede comprobar, el porcentaje de acierto en la clasificación del parámetro acción es cercano a un 70%, porcentaje superado por el conseguido al modelar el agente CMUnited 99. Esto puede implicar que el agente a etiquetar muestre una cierta imprevisibilidad, dificultando su estudio.

Los índices de correlación muestran resultados discretos, obteniendo una correlación relativamente baja en la clasificación del ángulo de giro del agente rival.

4.2.4.2 CREACIÓN DEL MODELADOR

En la Tabla 14 se pueden observar los resultados de la creación del **modelador** en el agente YowAI.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	69,05%
Turn angle	0,614
Dash power	0,788

Tabla 14: Resultados de la creación del modelador para el agente YowAI

Al crear el modelador se observa que los valores han mejorado perceptiblemente, sobre todo los índices de correlación.

4.2.4.3 RESULTADOS DE LA SIMULACIÓN

La Tabla 15 y la Tabla 16 muestran información de los resultados obtenidos al realizar las simulaciones con y sin modelo del agente YowAI, empezando por la **simulación sin modelo**, es decir, la media de las jugadas realizadas entre el delantero por defecto y el portero YowAI.

PARÁMETRO	VALOR OBTENIDO
Goles	8,87
Paradas portero	16,17
Tiros fuera	4,89
Fallos de simulación	3 fallos

Tabla 15: Resultados de la simulación sin modelo del agente YowAI

A continuación en la Tabla 16 se muestran los resultados de la **simulación al añadir el modelo generado al agente CMUnited 99**, es decir, la media de las jugadas realizadas entre el delantero al que se le ha añadido el modelo del rival y el portero YowAI.

PARÁMETRO	VALOR OBTENIDO
Goles	9,84
Paradas portero	14,07
Tiros fuera	6,06
Fallos de simulación	0 fallos

Tabla 16: Resultados de la simulación con modelo del agente YowAI

Como era de esperar se ha producido un aumento en la tasa de goles al añadir el modelo, concretamente de un gol más por cada iteración de treinta jugadas. Así mismo se observa que la tasa de fallos, referida a los tiros que han sido repelidos por los postes, ha disminuido de tres a cero fallos, mientras que los tiros fuera también han aumentado.

4.2.4.4 GANANCIA EN LA SIMULACIÓN POR LA INCLUSIÓN DEL MODELO

La Tabla 17 muestra la ganancia observada en la simulación.

PARÁMETRO	GANANCIA NETA	GANANCIA PORCENTUAL
Goles	+0,97 goles	+10,94%
Paradas del portero	-2,1 paradas	-12,99%

Tabla 17: Ganancia en la simulación debida a la inclusión del modelo en el agente YowAI

Comprobando los resultados obtenidos, se muestra una mejoría en el porcentaje de goles de casi un 11%. Esta mejoría no es tan evidente como la obtenida contra el agente UvA Trilearn, pero continúa siendo significativa.

4.2.5 MODELADO DEL AGENTE VIRTUAL WERDER

El siguiente agente portero sobre el que se realizarán pruebas es el agente Virtual Werder 2004. Este agente presenta una habilidad mediocre como portero, mostrando varios defectos en su construcción que son fácilmente aprovechados por cualquier agente delantero. Su programación le hace reaccionar adelantando su posición ante la cercanía del rival, lo que en principio debería resultar ventajoso por reducir los ángulos de visión de la portería del agente delantero, pero su lentitud de movimientos le hace vulnerable al ceder siempre una opción clara al delantero, puesto que se adelanta de tal forma que protege menos uno de sus palos.

Los siguientes puntos muestran los resultados obtenidos en la creación del etiquetador, en la generación del modelador, y los resultados de la simulación, además de una comparativa de la ganancia obtenida:

4.2.5.1 CREACIÓN DEL ETIQUETADOR

En la Tabla 18 se pueden observar los resultados de la creación del **etiquetador** en el agente Virtual Werder.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	83,57%
Turn angle	0,305
Dash power	0,387

Tabla 18: Resultados de la creación del etiquetador para el agente Virtual Werder

Como se muestra en la tabla superior, los resultados de la creación del etiquetador muestran un porcentaje de clasificación de la acción ligeramente superior al resto. Así mismo el coeficiente de correlación del ángulo de giro es el mayor de todos los agentes estudiados.

4.2.5.2 CREACIÓN DEL MODELADOR

En la Tabla 19 se muestran los resultados de la creación del **modelador** en el agente Virtual Werder.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	86,74%
Turn angle	0,655
Dash power	0,414

Tabla 19: Resultados de la creación del modelador para el agente Virtual Werder

Una vez integrado el etiquetador los índices han experimentado una mejoría notable, consiguiendo el porcentaje de clasificación de la acción más alto de todos los porteros clasificados.

4.2.5.3 RESULTADOS DE LA SIMULACIÓN

La Tabla 20 y la Tabla 21 muestran información de los resultados obtenidos al realizar las simulaciones con y sin modelo del agente Virtual Werder, empezando por la **simulación sin modelo**, es decir, la media de las jugadas realizadas entre el delantero por defecto y el portero Virtual Werder.

PARÁMETRO	VALOR OBTENIDO
Goles	20,07
Paradas portero	2,03
Tiros fuera	7,81
Fallos de simulación	2 fallos

Tabla 20: Resultados de la simulación sin modelo del agente Virtual Werder

A continuación la Tabla 21 muestra los resultados de la **simulación al añadir el modelo generado al agente CMUnited 99**, es decir, la media de las jugadas realizadas entre el delantero al que se le ha añadido el modelo del rival y el portero Virtual Werder.

PARÁMETRO	VALOR OBTENIDO
Goles	21,03
Paradas portero	2,21
Tiros fuera	6,63
Fallos de simulación	5 fallos

Tabla 21: Resultados de la simulación con modelo del agente Virtual Werder

Al observar la tasa de goles marcados se comprueba que como era de esperar no se anotan muchos más goles al añadir el modelo al agente delantero, puesto que dos tercios de los tiros ya eran goles. De todas formas se ha conseguido una pequeña mejoría con la aplicación del modelo.

4.2.5.4 GANANCIA EN LA SIMULACIÓN POR LA INCLUSIÓN DEL MODELO

La Tabla 22 muestra la ganancia observada en la simulación.

PARÁMETRO	GANANCIA NETA	GANANCIA PORCENTUAL
Goles	+0,96 goles	+4,78%
Paradas del portero	+0,18 paradas	+8,87%

Tabla 22: Ganancia en la simulación debida a la inclusión del modelo en el agente Virtual Werder

En resumidas cuentas, la ganancia obtenida es de alrededor de un 5%. En el caso del agente UvA Trilearn la mejora fue de un 200%, y en el caso del agente YowAI fue de un 10%, por lo que la ganancia obtenida con el agente Virtual Werder no es tanta como la obtenida con los otros agentes. Esta mejora entra dentro de lo esperado debido a la falta de habilidad del portero.

Así mismo se muestra por primera vez en la simulación un incremento de los tiros que salen fuera de la portería, es decir, los tiros que ni acaban siendo gol ni los que acaban siendo parados por el portero. Esta ganancia se podría explicar con la necesidad de apurar más los tiros para asegurar el gol.

4.2.6 MODELADO DEL AGENTE WRIGHT EAGLE

El siguiente portero objeto de estudio es el agente Wright Eagle 2005. Este agente presenta una tasa de goles encajados media-alta, situándose entre los agentes YowAI y Virtual Werder.

Los siguientes puntos muestran los resultados obtenidos en la creación del etiquetador, en la generación del modelador, y los resultados de la simulación, además de una comparativa de la ganancia obtenida.

4.2.6.1 CREACIÓN DEL ETIQUETADOR

En la Tabla 23 se muestran los resultados de la creación del **etiquetador** en el agente Wright Eagle.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	69.50 %
Turn angle	0.042
Dash power	0.285

Tabla 23: Resultados de la creación del etiquetador para el agente Wright Eagle

Como se puede comprobar, el porcentaje de acierto en la clasificación del parámetro acción es cercano a un 70%, y los índices de correlación muestran resultados cercanos al 0. Estos valores indican que los conjuntos de clasificadores obtenidos no han conseguido una buena clasificación de las muestras.

4.2.6.2 CREACIÓN DEL MODELADOR

En la Tabla 24 se pueden observar los resultados de la creación del **modelador** en el agente Wright Eagle.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	73.46 %
Turn angle	0.756
Dash power	0.494

Tabla 24: Resultados de la creación del modelador para el agente Wright Eagle

Al crear el modelador se observa que los valores han mejorado significativamente, sobre todo los índices de correlación.

4.2.6.3 RESULTADOS DE LA SIMULACIÓN

La Tabla 25 y la Tabla 26 muestran información de los resultados obtenidos al realizar las simulaciones con y sin modelo del agente Wright Eagle, empezando por la **simulación sin modelo**, es decir, la media de las jugadas realizadas entre el delantero por defecto y el portero Wright Eagle.

PARÁMETRO	VALOR OBTENIDO
Goles	16.1
Paradas portero	7.9
Tiros fuera	5.9
Fallos de simulación	5 fallos

Tabla 25: Resultados de la simulación sin modelo del agente Wright Eagle

A continuación en la Tabla 16 se muestran los resultados de la **simulación al añadir el modelo generado al agente CMUnited 99**, es decir, la media de las jugadas realizadas entre el delantero al que se le ha añadido el modelo del rival y el portero Wright Eagle.

PARÁMETRO	VALOR OBTENIDO
Goles	20.79
Paradas portero	4.48
Tiros fuera	4.64
Fallos de simulación	11 fallos

Tabla 26: Resultados de la simulación con modelo del agente Wright Eagle

En los resultados se puede observar que se ha producido un aumento de aproximadamente un tercio en la tasa de goles al añadir el modelo. El parámetro paradas del portero muestra que aproximadamente la mitad de los tiros, que antes eran detenidos por el portero, ahora acaban en gol. Cabe destacar el aumento de los fallos de simulación, siendo el agente Wright Eagle el que más tiros al palo recibe de todos los porteros observados, con y sin modelo añadido.

4.2.6.4 GANANCIA EN LA SIMULACIÓN POR LA INCLUSIÓN DEL MODELO

La Tabla 27 muestra la ganancia observada en la simulación.

PARÁMETRO	GANANCIA NETA	GANANCIA PORCENTUAL
Goles	+4,69 goles	+29.13%
Paradas del portero	-3,42 paradas	-43.29%

Tabla 27: Ganancia en la simulación debida a la inclusión del modelo en el agente Wright Eagle

Comprobando los resultados obtenidos, se muestra una mejoría muy significativa en el porcentaje de goles, alcanzando casi un 30% más.

4.2.7 MODELADO DEL AGENTE TOKIO TECH

El siguiente portero objeto de estudio es el agente Tokio Tech SFC 2005. Este agente presenta una tasa media de goles encajados, muy similar al portero YowAI, aunque los resultados obtenidos al incluir el modelo en el delantero son muy superiores a dicho agente.

Los siguientes puntos muestran los resultados obtenidos en la creación del etiquetador, en la generación del modelador, y los resultados de la simulación, además de una comparativa de la ganancia obtenida.

4.2.7.1 CREACIÓN DEL ETIQUETADOR

En la Tabla 28 se muestran los resultados de la creación del **etiquetador** en el agente Tokio Tech.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	69.78 %
Turn angle	0.027
Dash power	0.400

Tabla 28: Resultados de la creación del etiquetador para el agente Tokio Tech

Como se puede comprobar, el porcentaje de acierto en la clasificación del parámetro acción es cercano a un 70%. Destaca el índices de ángulo de giro, muy cercano al 0, que indica que los conjuntos de clasificadores obtenidos no han conseguido una buena clasificación de las muestras.

4.2.7.2 CREACIÓN DEL MODELADOR

En la Tabla 29 se pueden observar los resultados de la creación del **modelador** en el agente Tokio Tech.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	78.94 %
Turn angle	0.430
Dash power	0.549

Tabla 29: Resultados de la creación del modelador para el agente Tokio Tech

Al crear el modelador se observa que los valores han mejorado significativamente, alcanzando una clasificación del parámetro acción próxima al 80%, y un aumento del índice de correlación del ángulo de giro muy superior al obtenido en la creación del etiquetador.

4.2.7.3 RESULTADOS DE LA SIMULACIÓN

La Tabla 30 y la Tabla 31 muestran información de los resultados obtenidos al realizar las simulaciones con y sin modelo del agente Tokio Tech, empezando por la **simulación sin modelo**, es decir, la media de las jugadas realizadas entre el delantero por defecto y el portero Tokio Tech.

PARÁMETRO	VALOR OBTENIDO
Goles	10.21
Paradas portero	15.27
Tiros fuera	4.45
Fallos de simulación	1 fallo

Tabla 30: Resultados de la simulación sin modelo del agente Tokio Tech

A continuación en la Tabla 31 se muestran los resultados de la **simulación al añadir el modelo generado al agente CMUnited 99**, es decir, la media de las jugadas realizadas entre el delantero al que se le ha añadido el modelo del rival y el portero Tokio Tech.

PARÁMETRO	VALOR OBTENIDO
Goles	14.08
Paradas portero	11.72
Tiros fuera	4.1
Fallos de simulación	3 fallos

Tabla 31: Resultados de la simulación con modelo del agente Tokio Tech

En los resultados se puede observar que se ha producido un gran aumento en la tasa de goles al añadir el modelo, producto de que una cuarta parte de los tiros que antes eran detenidos por el portero ahora acaban siendo gol.

4.2.7.4 GANANCIA EN LA SIMULACIÓN POR LA INCLUSIÓN DEL MODELO

La Tabla 32 muestra la ganancia observada en la simulación.

PARÁMETRO	GANANCIA NETA	GANANCIA PORCENTUAL
Goles	+3.87 goles	+37.90%
Paradas del portero	-3,55 paradas	-23.25%

Tabla 32: Ganancia en la simulación debida a la inclusión del modelo en el agente Tokio Tech

Comprobando los resultados obtenidos, se muestra una mejoría cercana al 40% en el número de goles. Esta mejoría contrasta con la ganancia del 10% obtenida en el agente YowAI, puesto que ambos porteros tenían una tasa de goles similar sin modelo.

4.2.8 MODELADO DEL AGENTE FC PORTUGAL

El último portero objeto de estudio es el agente FC Portugal 2011, el más actual de todos los porteros observados. Este agente presenta una tasa media-baja de goles encajados, ofreciendo peores resultados que el agente UvA Trilearn, pero mejores que el resto de porteros.

Los siguientes puntos muestran los resultados obtenidos en la creación del etiquetador, en la generación del modelador, y los resultados de la simulación, además de una comparativa de la ganancia obtenida.

4.2.8.1 CREACIÓN DEL ETIQUETADOR

En la Tabla 33 se muestran los resultados de la creación del **etiquetador** en el agente FC Portugal.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	63.54 %
Turn angle	0.156
Dash power	0.162

Tabla 33: Resultados de la creación del etiquetador para el agente FC Portugal

Como se puede comprobar, el porcentaje de acierto en la clasificación del parámetro acción es ligeramente inferior a un 65%, y que los índices de correlación muestran resultados cercanos al 0. Estos valores indican que los conjuntos de clasificadores obtenidos no han conseguido una buena clasificación de las muestras.

4.2.8.2 CREACIÓN DEL MODELADOR

En la Tabla 34 se pueden observar los resultados de la creación del **modelador** en el agente FC Portugal.

TIPO DE CLASIFICADOR	VALOR OBTENIDO
Action	61.94 %
Turn angle	0.566
Dash power	0.670

Tabla 34: Resultados de la creación del modelador para el agente FC Portugal

Al crear el modelador se observa que los índices de correlación han mejorado significativamente, no así el clasificador de la acción que permanece estable.

4.2.8.3 RESULTADOS DE LA SIMULACIÓN

La Tabla 35 y la Tabla 36 muestran información de los resultados obtenidos al realizar las simulaciones con y sin modelo del agente FC Portugal, empezando por la **simulación sin**

modelo, es decir, la media de las jugadas realizadas entre el delantero por defecto y el portero FC Portugal.

PARÁMETRO	VALOR OBTENIDO
Goles	6.55
Paradas portero	19.29
Tiros fuera	4.05
Fallos de simulación	2 fallos

Tabla 35: Resultados de la simulación sin modelo del agente FC Portugal

A continuación en la Tabla 36 se muestran los resultados de la **simulación al añadir el modelo generado al agente CMUnited 99**, es decir, la media de las jugadas realizadas entre el delantero al que se le ha añadido el modelo del rival y el portero FC Portugal.

PARÁMETRO	VALOR OBTENIDO
Goles	7.19
Paradas portero	18.82
Tiros fuera	3.83
Fallos de simulación	0 fallos

Tabla 36: Resultados de la simulación con modelo del agente FC Portugal

En los resultados se puede observar que se ha producido un ligero aumento en la tasa de goles al añadir el modelo. Las paradas del portero y los tiros no varían significativamente, y además se muestra una reducción de los tiros al palo, no produciéndose ninguno al incluir el modelo.

4.2.8.4 GANANCIA EN LA SIMULACIÓN POR LA INCLUSIÓN DEL MODELO

La Tabla 37 muestra la ganancia observada en la simulación.

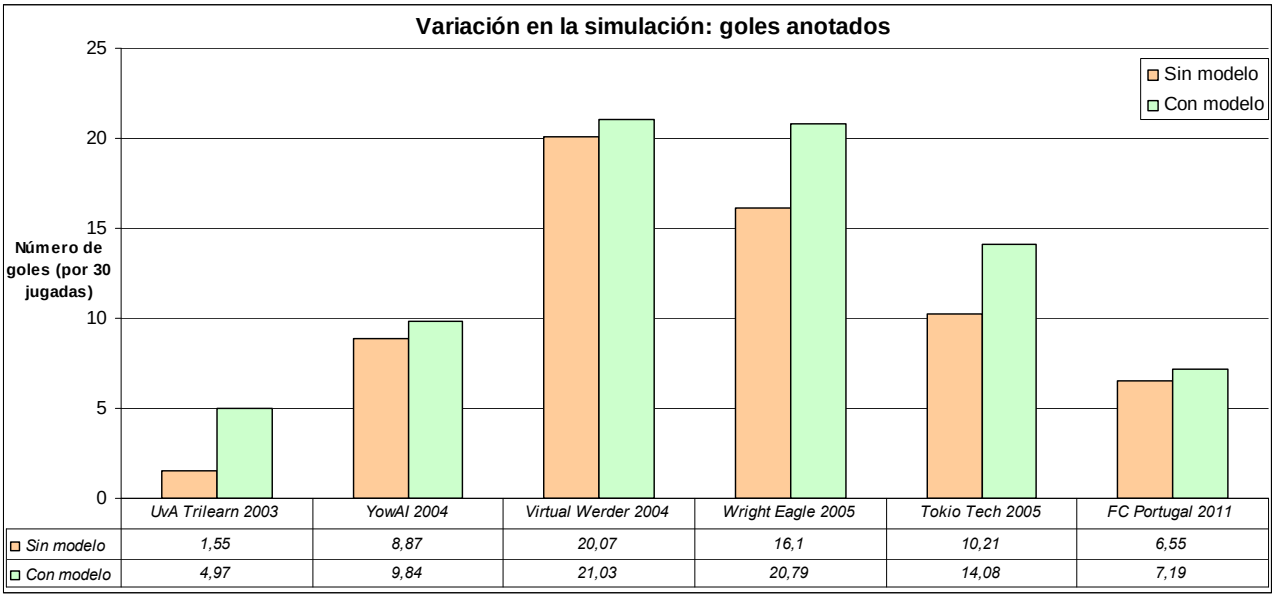
PARÁMETRO	GANANCIA NETA	GANANCIA PORCENTUAL
Goles	+0.64 goles	+9.77%
Paradas del portero	-0.47 paradas	-2.43%

Tabla 37: Ganancia en la simulación debida a la inclusión del modelo en el agente FC Portugal

Comprobando los resultados obtenidos, se muestra una ligera mejoría en el porcentaje de goles, anotando un 10% más al añadir el modelo. Esta mejora, aunque significativa, no es tan evidente como la de otros agentes.

4.2.9 COMPARATIVA DE RESULTADOS

En el gráfico mostrado en la Figura 17 se observa con claridad que, una vez aplicado el modelo del comportamiento del portero rival, se produce una mejora de los goles anotados. En los equipos de los años 2003 y 2004 se observa que cuantos más goles se anotan menor es la ganancia obtenida al añadir el modelo, mientras que en los equipos de 2005 y el equipo de 2011 no se muestra este comportamiento.



	UvA Trilearn 2003	YowAI 2004	Virtual Werder 2004	Wright Eagle 2005	Tokio Tech 2005	FC Portugal 2011
Sin modelo	1,55	8,87	20,07	16,1	10,21	6,55
Con modelo	4,97	9,84	21,03	20,79	14,08	7,19

Figura 17: Variación en la simulación: goles anotados

Un estudio de los resultados dictamina que hay varios niveles de habilidad en los porteros, llamando la atención que el agente UvA Trilearn encaja muy pocos goles en comparación con los demás. Además este equipo es el que experimenta una mayor mejora, del orden de 220% en comparación con el 40% de mejora que tiene el siguiente mejor equipo. La explicación de por qué un agente aumenta tanto sus resultados radica en que el agente UvA Trilearn es una evolución del agente CMUnited 99, por lo que el portero está previamente adaptado a ese delantero original. Una vez introducido el modelo en el delantero, el comportamiento de éste varía significativamente, por lo que el portero pierde esa adaptación que tenía anteriormente, igualándolo al resto de equipos. De este modo, una vez introducido el modelo, el agente UvA Trilearn sigue mostrando el menor número de goles encajados, pero la diferencia con el segundo mejor equipo, el FC Portugal, no es tan grande.

La Figura 18 muestra la variación en los tiros que han ido fuera, es decir que no han sido gol ni paradas del portero, en las simulaciones realizadas:

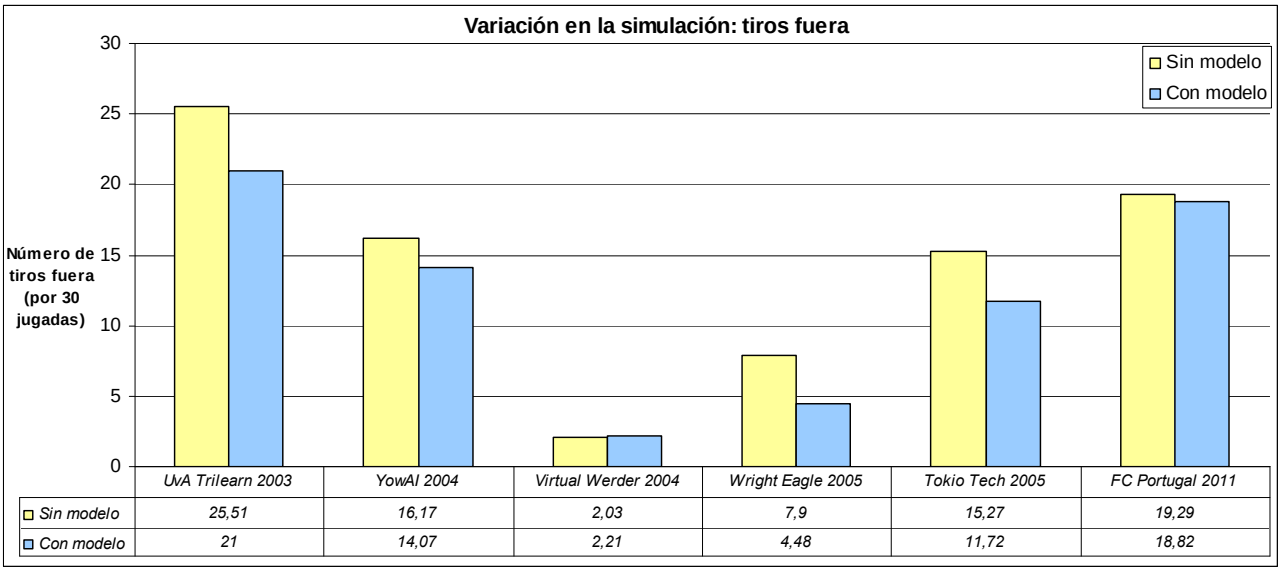


Figura 18: Variación en la simulación: tiros fuera

Como se puede observar, hay una cierta disminución en los tiros que se van fuera de la portería a la hora de incorporar el modelo en los porteros. Esta reducción de los tiros fuera es inversamente proporcional a la ganancia en el número de goles, salvo en el caso del agente Virtual Werder, en el que al ser la habilidad del portero tan reducida el número de balones que van fuera es un porcentaje tan pequeño que la inclusión o no del modelo es irrelevante.

4.2.10 CONCLUSIONES DE LA EXPERIMENTACIÓN

Tal y como ha quedado reflejado en las pruebas realizadas, se puede observar que el proceso de generación del etiquetador y del modelador y su posterior inclusión en el agente proporcionan una mejoría significativa en la adaptación de éste al entorno, mostrándose esta mejoría en un índice de goles anotados superior a los obtenidos cuando se realiza la misma simulación sin el modelo.

Esta mejoría viene determinada porque afortunadamente las acciones del agente rival se pueden clasificar con un porcentaje de acierto considerablemente alto, lo cual significa que el entorno escogido es determinista, puesto que si el comportamiento del agente rival fuera completamente aleatorio jamás se podría clasificar correctamente, y en consecuencia sería imposible adaptarse correctamente a dicho agente. Al no ser éste el caso, la utilización de técnicas de aprendizaje automático está completamente justificada.

En este caso el entorno escogido permite una medición directa de la optimización del agente a través de los goles marcados, lo cual conlleva a que los datos estadísticos hablen por sí solos.

Como se ha podido observar, se ha producido un incremento de los goles obtenidos en todos los casos. En el caso del agente UvA Trilearn se ha conseguido una mejoría que lleva a duplicar los goles anotados, mientras que en el resto de los casos la ganancia varía entre el cuarenta y el cinco por ciento de aumento en la anotación.

Esta diferencia en la mejora era previsible puesto que la variable de habilidad del portero tiene una relevancia enorme en las simulaciones, ya que cuanto menor sea la capacidad del portero de detener los disparos, menor será la posibilidad de pulir la técnica del delantero para mejorar la capacidad de anotar, ya que habrá un menor margen de mejora. Es decir, si el portero es muy malo, la mayoría de los disparos acabarán en gol independientemente de que el delantero tenga o no incorporado un modelo. Por otra parte, como se ha demostrado, si el portero es muy bueno y encaja pocos goles, entonces sí hay posibilidad de mejoría por parte del delantero. Este comportamiento, en el que la mejora es directamente proporcional a la habilidad del portero, se muestra en los equipos del año 2003 y 2004. En equipos posteriores, presumiblemente más complejos, este supuesto no es verificable.

Si se observan los datos de la experimentación, se ve que en las simulaciones con modelo las paradas del portero tienden a disminuir y los tiros fuera de la portería tienden a aumentar (mientras que los tiros al palo son más erráticos de controlar). Este hecho tiene muchas interpretaciones, pero la explicación más sencilla consisten en que el delantero trata de apurar más, buscando el límite de la portería para sortear al portero, con lo que aumentaría el porcentaje de éxito pero también el de balones tirados fuera, porque si el portero se adelanta a la jugada, está bien colocado, y es agresivo en sus movimientos, la única posibilidad que le queda al delantero será la de tirar el balón fuera.

Hay un dato interesante que se obtiene tras observar los resultados de las pruebas. El porcentaje de acierto en la clasificación de las acciones a la hora de crear el etiquetador y a la hora de crear el modelador aumenta una vez incorporado el etiquetador al agente delantero. Es decir, los coeficientes de correlación siempre son superiores al crear el modelador que al crear el etiquetador. Este comportamiento se debe a que el etiquetador implementado se encarga de generar los datos con los que aprende el modelador, por lo que el proceso se encuentra guiado.

Cabe destacar que durante la realización de las experimentaciones se reconoció a tiempo un error en el clasificador/etiquetador. En una comprobación de los ficheros de datos generados como entrada de la herramienta *Weka*, se descubrió que uno de los parámetros generados estaba incorrectamente calculado. Este parámetro en concreto era un atributo calculado que representaba el desplazamiento del jugador (en adelante *DESP_O*), que se calculaba como la distancia euclídea de su posición en los ejes de coordenadas, y resultó que influía de forma importante en la clasificación, obteniendo mejores resultados. Los siguientes datos muestran la variación obtenida:

A la hora de crear el etiquetador:

Antiguo parámetro *DESP_O*:

- *Dash power*: coeficiente de correlación 0.5791
- *Turn angle*: coeficiente de correlación 0.2094
- Acción: instancias correctamente clasificadas 76.458 %

Nuevo parámetro *DESP_O*:

- *Dash power* (nuevo *DESP_O*): coeficiente de correlación 0.597
- *Turn angle* (nuevo *DESP_O*): coeficiente de correlación 0.206
- Acción (nuevo *DESP_O*): instancias correctamente clasificadas 77.2942%

Tal y como se ve, a la hora de generar el etiquetador casi no hay diferencias apreciables.

A la hora de crear el modelador:

Antiguo parámetro *DESP_O*:

- *Dash power* (antiguo *DESP_O*): coeficiente de correlación 0.7216
- *Turn angle* (antiguo *DESP_O*): coeficiente de correlación 0.4643
- Acción (antiguo *DESP_O*): instancias correctamente clasificadas 79.2885%

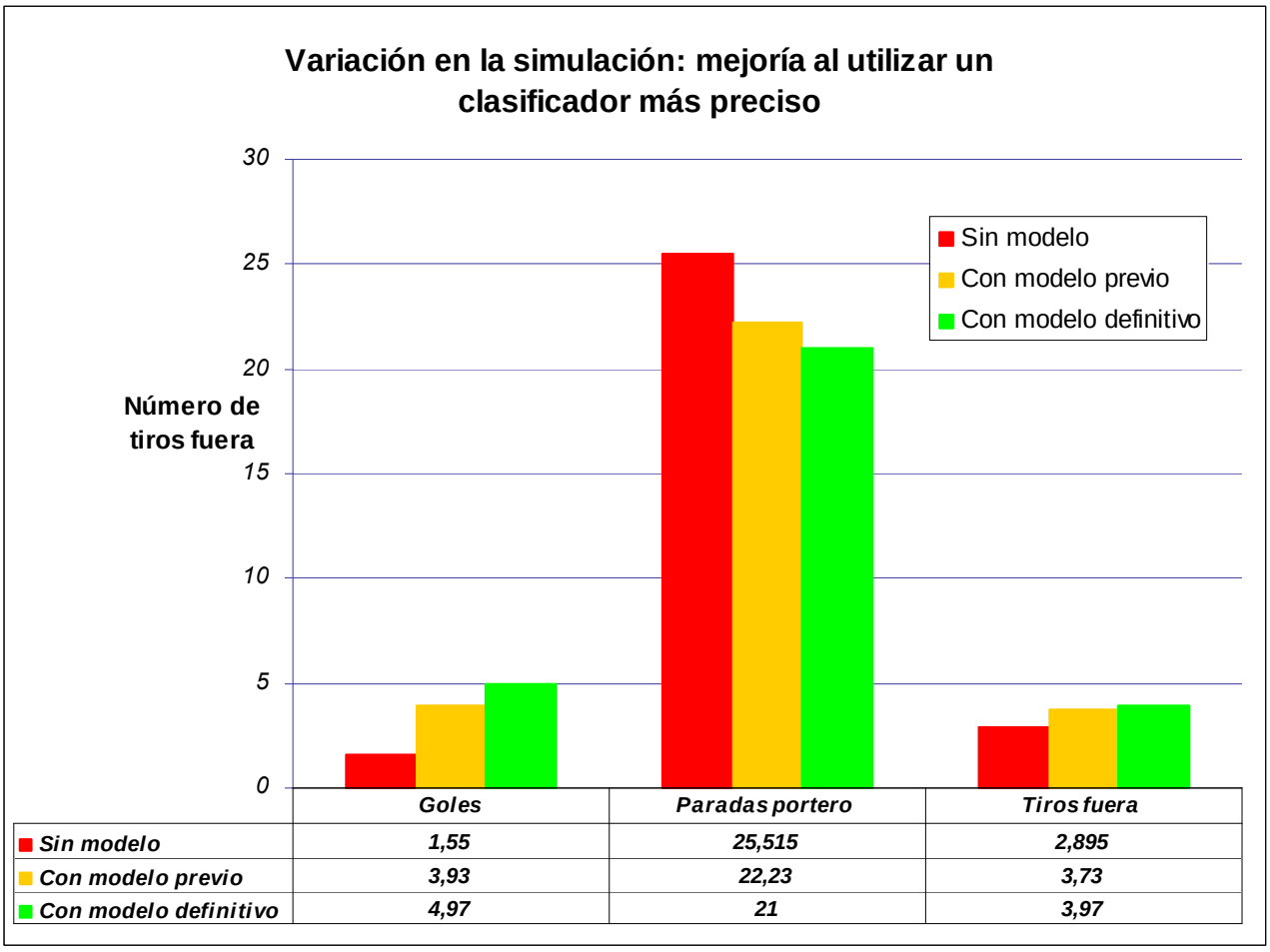
Antiguo parámetro *DESP_O*:

- *Dash power* (nuevo *DESP_O*): coeficiente de correlación 0.8378
- *Turn angle* (nuevo *DESP_O*): coeficiente de correlación 0.6225

- Acción (nuevo *DESP_O*): instancias correctamente clasificadas 78.9726%

En cambio a la hora de generar el modelo, que será incorporado al agente, sí que hay diferencias significativas. Se observa una mejora muy importante en los clasificadores basados en las reglas M5, mejorando la clasificación en el *Turn angle* en casi dos décimas.

La Figura 19 muestra los resultados comparando el antiguo modelador y el nuevo con la simulación sin modelo, siempre contra el agente UvA Trilearn:



	Goles	Paradas portero	Tiros fuera
Sin modelo	1,55	25,515	2,895
Con modelo previo	3,93	22,23	3,73
Con modelo definitivo	4,97	21	3,97

Figura 19: Variación en la simulación: mejoría al utilizar un clasificador más preciso

Se puede observar que la mejoría en la clasificación también conlleva una mejoría en el comportamiento global del delantero, metiendo más goles cuanto más preciso es el modelo, con lo que se ha afinado la habilidad del delantero de manera importante. Esto lleva a pensar que puede que varios parámetros del clasificador sean innecesarios, y también que puede haber otros parámetros que no se han tenido en cuenta y que podrían ayudar en la clasificación, utilizando por supuesto operaciones no permitidas por los clasificadores escogidos, como potencias de parámetros u operaciones sinusoidales, aunque no está claro que vayan a servir para mejorar la clasificación.

5 CONCLUSIONES GENERALES

La intención con la que nació este trabajo fue la de continuar con el trabajo de la tesis doctoral "Aprendizaje automático en conjuntos de clasificadores heterogéneos y modelado de agentes" [3], tarea que se ha realizado en la medida de lo posible. El ámbito del aprendizaje automático y el modelado de agentes dan para mucho más que este proyecto, como se puede observar en las líneas de trabajo futuras.

El objetivo final de este proyecto ha sido poner a disposición de futuros desarrolladores un entorno completo en el que realizar todos los pasos de forma automática (sin necesidad de realizar operaciones adicionales ni estar modificando el código continuamente) que permitan comprobar que la generación del modelo de comportamiento del agente rival es válido y utilizable, lo cual se ha conseguido, proporcionando además ejemplos cuantificables del éxito de esta tarea.

Hay que reconocer que la falta de documentación del entorno de la RoboCup dificultó el proceso en sus inicios, puesto que los problemas surgidos a la hora de instalar y configurar las herramientas eran constantes, con múltiples incompatibilidades que se fueron arreglando poco a poco. En esta línea se encuentran el agente delantero, que es una modificación del agente ganador de la competición del año 1999, que cuenta con cierta documentación pero que en ningún caso es completa, y lo mismo ocurre con el agente *Trainer*. Afortunadamente la existencia de herramientas de generación de documentación automática y un exhaustivo estudio de los diferentes ficheros de los que se componían facilitaron las cosas a la hora de realizar modificaciones en ambos casos.

Una vez terminados todos los programas y realizadas las optimizaciones, se ha conseguido una sensación de coherencia y estabilidad que por sí solas merecen la pena todo el trabajo realizado. Además, es probable que la generación de modelos de comportamientos ayude de algún modo a alcanzar la no tan lejana meta de la RoboCup, ésa que dice que para el año 2050 se podrá crear un equipo de robots humanoides que ganaría al mejor equipo humano jugando al fútbol.

6 FUTURAS LÍNEAS DE DESARROLLO

Se ha conseguido establecer una serie de pasos a seguir que facilitan la tarea para futuras líneas de trabajo, aunque por supuesto este proceso es ampliable y mejorable.

Al ser el entorno de la RoboCup fácilmente escalable, el siguiente paso inmediato en la simulación consistiría en añadir un defensa rival al entorno en el que se hacen las simulaciones. Esto aportaría nuevos datos y obligaría a replantearse las estrategias del delantero, puesto que no es lo mismo simular basándose en un partido de uno contra uno que en un partido con más jugadores. Los resultados serían peores, disminuyendo el número de goles de manera clara, como dicta la lógica. También se podrían añadir más delanteros, para posibilitar la creación de jugadas entre ellos dependiendo del comportamiento de el/los defensa/s y el portero, hasta completar un modelo con todos los jugadores. Para elaborar este modelo habría que modificar y ampliar los atributos con los que se realiza el clasificador, con lo que aumentaría la complejidad de éste, y no es seguro que en entornos completos llegara a funcionar correctamente, puesto que cuando entran tantas variables en juego es complicado establecer patrones de comportamientos globales. Quizá un enfoque hacia el modelado del portero y los agentes defensas sea más que suficiente.

Sobre el conjunto de los programas realizados, una forma sencilla de continuar sería agrupar los programas en la medida de lo posible, bien juntando en una misma aplicación aquellos que utilicen como entrada salidas de otros, o bien preparando una suite que de alguna manera automatice este proceso haciéndolo transparente al usuario, requiriendo su atención únicamente para las diferentes opciones que se ejecuten (bien sea etiquetar o modelar, o si se desea obtener algún fichero intermedio en particular, como puede ser el archivo de entrada de *Weka* para un determinado clasificador, por ejemplo).

Una posible línea de desarrollo futura es la utilización de las próximas versiones de los programas que aporte la RoboCup, puesto que es una iniciativa que está en continuo proceso de evolución. En principio, y a no ser que se hagan cambios estructurales profundos, los agentes utilizados en este proyecto serán compatibles con todas las futuras versiones que se vayan desarrollando, puesto que la mayoría de las actualizaciones corrigen fallos leves y añaden nuevos parámetros opcionales. Tal como está programado, tanto el agente delantero como el *Trainer* cargan el fichero de parámetros del servidor, y se da el caso de que muchos de estos parámetros son posteriores al año de creación del agente, con lo que se ha tomado la decisión de diseño de recogerlos y almacenarlos para su posterior utilización, con lo que están disponibles para futuros trabajos.

7 BIBLIOGRAFÍA

[1] A. Bandura. *Social Learning Theory*, año 1977.

[2] P. Riley y M. Veloso. *On behavior classification in adversarial environments*, año 2000.

[3] Agapito Ledezma. Aprendizaje automático en conjuntos de clasificadores heterogéneos y modelado de agentes, año 2004

[4] M.J. Wooldridge y N.R. Jennings. *Intelligent agents: Theory and practice. Knowledge Engineering Review*, año 1995.

[5] Charles M. Macal y Michael J. North. *Agent-based modeling and simulation: ABMS examples*, año 2008.

[6] K. P. Sycara. *Multiagent systems*, año 1998.

[7] Gal A. Kaminka, Milind Tambe, C.M. Hopper. *The role of agent modelling in agent robustness*, año 1998.

[8] Peter Stone, P. Riley, and M. Veloso. *Defining and using ideal teammate and opponent models*. Conferencia IAAI, año 2000.

[9] T. Steffens. *Adapting similarity measures to agent types in opponent modelling*, año 2004.

[10] H. Kautz y J. Allen. *Generalized plan recognition*. Conferencia AAAI, año 1986.

[11] L.E. Baum y T. Petrie. *Statistical Inference for Probabilistic Functions of Finite State Markov Chains*, año 1966.

[12] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*, año 1988.

[13] P. Riley y M. Veloso. *Planning for distributed execution through use of probabilistic opponent models*. En Procedimientos de la 6ª conferencia internacional de planificación de IA (AIPS-2002), año 2002.

[14] Les Gasser. *MAS infraestructure definitions, needs and prospects*, año 2000.

[15] Mao Chen† , Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang†, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang †, Xiang Ying †. *Users Manual, RoboCup Soccer Server for Soccer Server Version 7.07 and later*, año 2002.

[16] Itsuki Noda, Peter Stone. *Autonomous Agents and Multi-Agent Systems*, año 2003.

[17] Ethem Alpaydin. *Introduction to Machine Learning*, año 2004.

[18] Manuela Veloso, Enrico Pagello, Hiroaki Kitano. *RoboCup-99: Robot Soccer World Cup III (Lecture Notes in Computer Science / Lecture Notes in Artificial Intelligence)*, año 2000.

[19] Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, Tomoichi Takahashi. *RoboCup 2006: Robot Soccer World Cup X (Lecture Notes in Computer Science)*, año 2007.

[20] Google. (Buscador web) www.google.com

[21] Wikipedia (Enciclopedia on-line) www.wikipedia.org

[22] Página oficial de preguntas frecuentes de Google Bot http://www.google.com/bot.html

[23] Página oficial de preguntas frecuentes de Yahoo! Slurp http://help.yahoo.com/l/us/yahoo/search/webcrawler/

[24] Página oficial del proyecto Coriolis http://www.coriolis.eu.org/

[25] Página oficial del aspirador Roomba http://www.irobot.com/

[26] Página oficial de RoboCup http://www.robocup.org/

[27] Página oficial de la RoboCup 2011 celebrada en Estambul http://www.robocup2011.org/en/

[28] Reglas RoboCup categoría Humanoide http://www.tzi.de/humanoid/pub/Website/Downloads/HumanoidLeagueRules2010.pdf

[29] Página oficial del equipo Team Osaka http://www.vstone.co.jp/top/products/robot/v2/press/index.html

[30] Reglas RoboCup categoría Estándar http://www.tzi.de/spl/pub/Website/Downloads/Rules2011.pdf

[31] Página oficial del robot Nao http://www.aldebaran-robotics.com/eng/NaoRobocup.php

[32] Página oficial del robot Aibo http://support.sony-europe.com/aibo/

[33] Reglas RoboCup categoría *Medium Size* http://robocupmsl.googlegroups.com/web/msl-rules-2010-12-31.pdf?gda=XTe7zEsAAABO910EAeTjDfSefr1HKLQyd7jCg8ZDAmcumFgGYcYdr64UrsWbkEiSGdBj3YNeVEa6SfRaYegFJlGKsdVIL0HdBkXa90K8pT5MNmkW1w_4BQ

[34] Reglas RoboCup categoría *Small Size* http://small-size.informatik.uni-bremen.de/_media/rules:ssl-rules-2011.pdf

[35] Página principal de la categoría de RoboCupRescue de la competición RoboCup 2010 http://www.robocup2010.org/competition_Category.php?c=2

[36] Página del proyecto *GNU Lesser general public license* http://www.gnu.org/copyleft/lesser.html

[37] Página oficial del equipo CMUnited 99 http://www.cs.cmu.edu/~pstone/RoboCup/CMUnited99-sim.html

[38] Página oficial del equipo UvA Trilearn 2003 http://staff.science.uva.nl/~jellekok/robocup/2003/

[39] Página oficial del equipo Yow AI 2004 http://ne.cs.uec.ac.jp/~newone/YowAI2004.html

[40] Página oficial del proyecto Virtual Werder de la universidad de Bremen http://www.tzi.de/en/about-us/projects/?tx_tziprojects_pi1[project_id]=478

[41] Página oficial del laboratorio de sistemas multiagente de la Universidad de Ciencias y Tecnología de China http://wrighteagle.org/en/robocup/index.php

[42] Página oficial del Instituto Tecnológico de Tokio http://www.titech.ac.jp/english/

[43] Página oficial del Laboratorio de Inteligencia Artificial y Ciencias de la Computación de la Universidad de Oporto http://www.liacc.up.pt/

[44] Página oficial del Instituto de Ingeniería Electrónica y Telemática de la Universidad de Aveiro http://wiki.ieeta.pt/wiki/index.php/Main_Page

[45] Página oficial del proyecto Ubuntu http://www.ubuntu-es.org/

[46] Página oficial del proyecto Linux Mint http://linuxmint.com/

[47] Descargas de la RoboCup de la categoría de simulación 2D http://sourceforge.net/project/showfiles.php?group_id=24184

[48] Listas de distribución de la sección de simulación de la RoboCup https://lists.cc.gatech.edu/

[49] Página del equipo TokioTech http://rctools.sourceforge.jp/akiyama

[50] Página oficial de *Weka* http://www.cs.waikato.ac.nz/ml/weka/

Anexo A PLANIFICACIÓN Y PRESUPUESTO

El presente capítulo proporciona una visión general de las fases del proyecto a lo largo del tiempo de desarrollo, así como de los costes asociados que se han presentado durante el desarrollo.

A.1 PLANIFICACIÓN

Para llevar a cabo el desarrollo del proyecto se ha procedido a establecer una serie de tareas que marcan las diferentes fases del proyecto. Estas tareas son las siguientes:

- **Fase de análisis**: primera toma de contacto con el proyecto y con el dominio que abarca.
    - Propuesta del proyecto: fase en la que se forma una visión preliminar del proyecto a desarrollar.
    - Especificación de funcionalidades: fase en la que se especifican los requisitos que definen la funcionalidad del proyecto.
    - Estudios preliminares: estudio del estado del arte en general y del sistema de agentes inteligentes autónomos en particular.
    - Estudio de la iniciativa RoboCup y del agente CMUnited 99.
- **Fase de diseño**: estudio del trabajo inicial de Ledezma [3] y de las distintas maneras de desarrollar las diferentes aplicaciones necesarias.
- **Fase de implementación**: implementación de las diferentes aplicaciones generadas.
- **Pruebas**: fase de testeo y verificación de la validez del proyecto desarrollado.
    - Estudio previo clientes disponibles: obtención de agentes de tipo portero contra los que realizar la experimentación.
    - Experimentación propiamente dicha, en la que se obtienen los resultados tangibles.
- **Documentación**: creación de este documento, así como la documentación exhaustiva del código fuente generado en las diferentes aplicaciones desarrolladas.

La Figura 20 y su continuación, la Figura 21, muestran el diagrama de Gantt correspondiente a la planificación de este proyecto. El diagrama de Gantt es una representación gráfica empleada para mostrar el esfuerzo temporal necesario para llevar a cabo una tarea. En él se aprecia el tiempo empleado en cada una de las fases del proyecto.

La Figura 23 muestra la planificación en los cinco primeros meses. En la parte izquierda se pueden observar las diferentes tareas de las que se compone el desarrollo del proyecto, junto con su duración.

	Nombre de tarea	Duración
1	Inicio del proyecto	1 día
2	**Fase de análisis**	**54 días**
3	Propuesta del proyecto	8 días
4	Especificación de funcionalidades	9 días
5	Estudios preliminares	14 días
6	Estudio de la Robocup	21 días
7	**Fase de diseño**	**20 días**
8	Diseño de las diferentes aplicaciones	20 días
9	**Fase de implementación**	**49 días**
10	Implementación de las diferentes aplicaciones	49 días
11	**Pruebas**	**49 días**
12	Estudio previo clientes disponibles	21 días
13	Experimentación	28 días
14	Documentación	191 días
15	Fin del proyecto	1 día

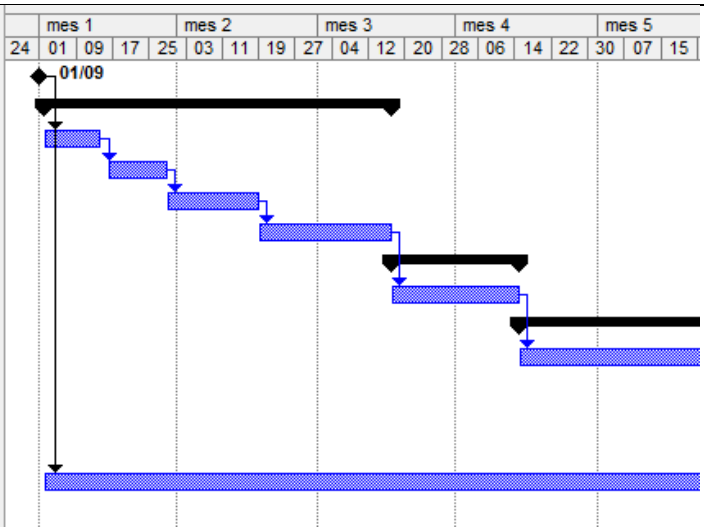


Figura 20: Planificación del proyecto: diagrama de Gantt parte I

La Figura 21 muestra el resto de la planificación, desde mediados del quinto mes hasta su finalización. Se ha optado por continuar mostrando el nombre de las tareas y su duración para aumentar la legibilidad.

	Nombre de tarea	Duración
1	Inicio del proyecto	1 día
2	**Fase de análisis**	**54 días**
3	Propuesta del proyecto	8 días
4	Especificación de funcionalidades	9 días
5	Estudios preliminares	14 días
6	Estudio de la Robocup	21 días
7	**Fase de diseño**	**20 días**
8	Diseño de las diferentes aplicaciones	20 días
9	**Fase de implementación**	**49 días**
10	Implementación de las diferentes aplicaciones	49 días
11	**Pruebas**	**49 días**
12	Estudio previo clientes disponibles	21 días
13	Experimentación	28 días
14	Documentación	191 días
15	Fin del proyecto	1 día

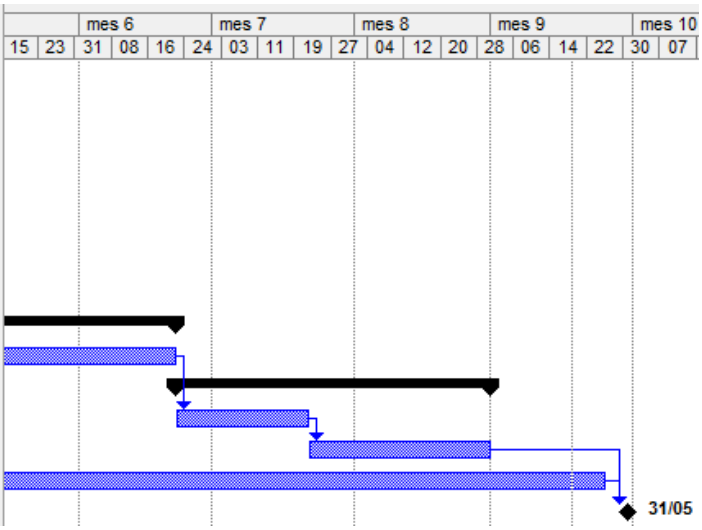


Figura 21: Planificación del proyecto: diagrama de Gantt parte II

El diagrama muestra una planificación a lo largo de nueve meses. Esta planificación no se ajusta estrictamente a la realidad, puesto que por motivos laborables se han producido largos períodos en los que no se ha realizado ninguna tarea. Así pues se ha optado por una representación ajustada al tiempo efectivo empleado en el desarrollo del proyecto.

A.2 PRESUPUESTO

En este apartado se detalla el cómputo del coste del proyecto, especificando tanto el gasto de personal como los gastos relacionados con el software, hardware y material fungible utilizado durante el proceso de desarrollo.

A.2.1 DESGLOSE POR ACTIVIDADES

El calendario laboral mostrado en el punto anterior toma como referencia una jornada laboral de cuatro horas diarias. Una planificación lo suficientemente precisa tendría diferentes niveles de carga de trabajo a lo largo del proceso, puesto que habría períodos de inactividad asociados a vacaciones, o períodos de máxima carga al ir finalizando los diferentes hitos. Como se ha explicado en el punto anterior, el proceso de desarrollo del proyecto ha abarcado más de esos nueve meses estipulados, así que la planificación refleja la media estimada a través del total de las horas empleadas.

La distribución de horas empleadas entre las diferentes fases es la siguiente:

- **Fase de análisis**: 54 días * 3 horas/día = 162 horas
- **Fase de diseño**: 20 días * 3 horas/día = 60 horas
- **Fase de implementación**: 49 días * 3 horas/día = 147 horas
- **Fase de pruebas**: 49 días * 3 horas/día = 147 horas
- **Fase de documentación**: 172 días * 1 hora/día + 19 días * 4 horas/día = 248 horas

Así pues el total de tiempo empleado entre todas las fases del proyecto asciende a 764 horas, distribuidas en un período de alrededor de nueve meses.

A.2.2 COSTE DE PERSONAL

A partir de las horas establecidas en el punto anterior, se realiza el cálculo de costes asociados al personal. Se ha optado por utilizar distintos perfiles de trabajador para cada una de las actividades de las que se compone el proyecto, concretamente se ha optado por contar con un analista para las fases de análisis y de diseño, un ingeniero para la fase de implementación y pruebas, y un responsable de documentación para la fase de documentación.

La Tabla 38 muestra el coste asociado al personal, tomando como referencias salarios de empresas similares del mismo sector. Al tener una jornada laboral de 4 horas al día, el valor de la dedicación hombre/mes corresponde a 80 horas de trabajo. Todos los sueldos son calculados sin aplicar el I.V.A.

PERFIL	Nº DE HORAS	COSTE HORA	DEDICACIÓN (HOMBRE/MES)	COSTE HOMBRE/MES	TOTAL
Analista	222	35 €	2,775	2800 €	7770 €
Ingeniero	294	25 €	3,675	2200 €	7350 €
Responsable de documentación	248	15 €	3,1	1200 €	3720 €
					18840 €

Tabla 38: Coste de personal

A.2.3 COSTE DE HARDWARE

La Tabla 39 muestra el coste asociado al hardware utilizado durante el desarrollo del proyecto, junto con su cuadro de amortización. Dicho cálculo es realizado acorde a la plantilla de presupuesto proporcionada por la universidad Carlos III de Madrid descrita en el punto A.2.7.

DESCRIPCIÓN	COSTE	% DE USO DEDICADO	DEDICACIÓN MESES	PERÍODO DE DEPRECIACIÓN	COSTE IMPUTABLE
Acer Aspire One 110	349 €	100 %	9,55	60	55,55 €
Ratón y teclado Microsoft	59 €	100 %	9,55	60	9,39 €
Monitor Asus LCD PW-201	179 €	100 %	9,55	60	28,49 €
HDD usb Western Digital 512 GB	99 €	100 %	9,55	60	15,76 €
Impresora HP PSC 1200	129 €	100 %	9,55	60	20,53 €
					129,72 €

Tabla 39: Coste de hardware

A.2.4 COSTE DE SOFTWARE Y LICENCIAS

La Tabla 40 muestra el coste asociado a las herramientas software utilizadas durante el desarrollo del proyecto. Los costes no incluyen el I.V.A.

DESCRIPCIÓN	COSTE IMPUTABLE
Microsoft Office 2007 Professional	289 €
Microsoft Office Project 2007	119 €
Weka	0 €
Eclipse Indigo	0 €
Plugin Doxygen para Eclipse	0 €
TeXnicCenter LaTeX IDE	0 €
	408 €

Tabla 40: Coste de software y licencias

A.2.5 COSTE DE MATERIAL FUNGIBLE

La Tabla 41 muestra el coste asociado al material fungible utilizado durante el desarrollo del proyecto. Los costes no incluyen el I.V.A.

DESCRIPCIÓN	COSTE IMPUTABLE
Material de oficina	60 €
Recambios de impresora	30 €
	90 €

Tabla 41: Coste de material fungible

A.2.6 RESUMEN DE COSTES

El siguiente paso consiste en realizar un resumen de todos los costes anteriormente imputados, es decir, costes de personal, costes de hardware, costes de software y licencias, y finalmente costes de material fungible. A la suma de todos estos costes se le aplica una tasa de costes indirectos fijada en un 20%, utilizando el valor que proporciona la plantilla de presupuesto proporcionada por la universidad Carlos III de Madrid descrita en el punto A.2.7. Además, una vez aplicada esta tasa, se ha de aplicar el impuesto sobre el valor añadido, que en el momento de realizar este proyecto se sitúa en el 18%. El presupuesto final asciende a 27.566,29 €.

La Tabla 42 muestra el resumen con todas estas tasas aplicadas.

DESCRIPCIÓN	COSTES TOTALES
Personal	18840 €
Amortización de equipos	129,72 €
Costes de funcionamiento	498 €
Total costes	19467,72 €
Costes indirectos (20% del total)	3893,54 €
Total costes sin IVA	23361,27 €
IVA 18%	4205,03 €
Total con IVA	**27566,29 €**

Tabla 42: Resumen de costes

Para finalizar el apartado de presupuesto, se procede a mostrar la plantilla de presupuesto, proporcionada por la universidad Carlos III de Madrid, ya rellenada con los datos del presupuesto.

A.2.7 PLANTILLA DE PRESUPUESTO

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Alberto Blanco Martínez

2.- Departamento:
Informática: Control, Aprendizaje y Optimización de Sistemas

3.- Descripción del Proyecto:
- Titulo: Optimización de la adquisición de modelos de oponentes en la robocup – categoría de simulación 2D
- Duración (meses): 9
Tasa de costes Indirectos: 20%

4.- Presupuesto total del Proyecto (valores en Euros):
27.566,29Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
Alberto Blanco Martínez		Analista	2,775	2.800,00	7.770,00	
Alberto Blanco Martínez		Ingeniero	3,675	2.000,00	7.350,00	
Alberto Blanco Martínez		Documentación	3,1	1.200,00	3.720,00	
		Hombres mes	**9,55**	**Total**	18.840,00	

[a] 1 Hombre mes = 80 horas

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Acer Aspire One 110	349,00	100	9,55	60	55,55
Ratón y teclado Microsoft	59,00	100	9,55	60	9,39
Monitor Asus LCD PW-201	179,00	100	9,55	60	28,49
HDD usb Western Digital 512 GB	99,00	100	9,55	60	15,76
Impresora HP PSC 1200	129,00	100	9,55	60	20,53
				Total	129,72

[d] Fórmula de cálculo de la Amortización:

$\frac{A}{B} \times C \times D$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
Microsoft Office 2007 Professional		289,00
Microsoft Office Project 2007		119,00
Material fungible		90,00
	Total	498,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	18.840,00
Amortización	129,72
Subcontratación de tareas	0,00
Costes de funcionamiento	498,00
Costes Indirectos	3.893,54
Total sin IVA	23.361,27
IVA 18%	4.205,03
Total	27.566,29

Figura 22: Plantilla de presupuesto

Anexo B MANUAL DE USUARIO

El entorno utilizado en este proyecto es fácilmente utilizable en otros equipos sin restricciones, puesto que todo el software requerido es libre bajo licencia GPL (*GNU public license*). Sin embargo hay que tener en cuenta los siguientes puntos:

B.1 REQUISITOS DEL SISTEMA

Los requisitos mínimos para funcionar son los siguientes:

- Sistema operativo Linux: para la realización de este proyecto se ha utilizado un sistema basado en Debian, concretamente la distribución Ubuntu [45], que se encuentra disponible bajo licencia GPL. El desarrollo comenzó con la versión *Dapper Drake*, y para la realización de las pruebas finales se ha optado por el paso a una nueva distribución de Linux basada en Debian, Linux Mint [46], de la que se usa la versión 11.0 llamada *Katya*. Así mismo, se ha comprobado su correcto funcionamiento en Ubuntu 12.04 *Precise Pangolin*. Existen versiones de la mayoría de los programas del entorno de la RoboCup para el sistema operativo Windows, pero no están actualizadas, por lo que no es aconsejable utilizarlo.
- Compilador GCC 4.5 o superior. Se recomienda instalarse el paquete *build-essential*, que contiene varias librerías de compilación.
- 512MB de memoria RAM: la herramienta *Weka* precisa más memoria dedicada de la que se asigna por defecto.
- Entorno de ejecución Java en su versión 1.4 o superior. Se requiere para ejecutar *Weka*.

El equipo de desarrollo ha sido un portátil con procesador Intel Centrino mononúcleo a 1.5 Ghz, con un 1 GB de memoria RAM. También se han realizado pruebas en un portátil con procesador Intel Atom a 1.5 Ghz con 1.5GB de memoria RAM.

B.2 CONTENIDO A INSTALAR

Se deben instalar los siguientes programas relacionados con el *Soccer Server*, todos ellos disponibles en la página web de la RoboCup Soccer Simulator [47]. Se utilizan las últimas versiones a fecha de Enero del año 2012:

- **Rcssbase**: *Soccer Base*, librerías base del *Soccer Server*. Al principio del desarrollo se utilizó la versión 10.0.11, aunque se ha asegurado la compatibilidad con la última versión disponible, la 12.1.3. Para que la instalación sea correcta, se ha de instalar la siguiente librería:
  - *Libboost* (librería C++ Boost). Versión actual 1.48.

- **Rcssserver:** el propio *Soccer Server*, el servidor en sí, encargado de soportar las ejecuciones y la comunicación entre agentes. Al igual que el base, se empezó con la versión 10.0.7, pero se ha asegurado la compatibilidad con la última versión, la 15.0.1. Ambas versiones se diferencian en que la más actual proporciona un sistema de gestión del juego más pulida, así como también corrige ciertos bugs, y dispone de nuevos mensajes para transmitir órdenes a los agentes, pero en lo que a este proyecto respecta no hay cambios visibles. Este programa presenta una serie de dependencias que hay que respetar para poder instalarlo:
  - Analizador léxico *Flex*. Versión actual 2.5.35.
  - Analizador léxico *Bison*. Versión actual 2.5.
- **Rcssmonitor**: *Soccer Monitor*, herramienta para visualizar los partidos en tiempo real. La versión utilizada es la 10.0.0, la versión actual es la 15.0.0. Para instalar esta herramienta se han de instalar las siguiente librerías:
  - *Libqt4* (librería de desarrollo de Qt). Versión actual 4.7.2.
  - *Libaudio-dev* (librería de desarrollo de audio). Versión actual 1.9.2-4
  - *Libxt-dev* (librería del entorno gráfico X11). Versión actual 1.0.9
  - *Libpng++-dev* (librería de desarrollo de png). Versión actual 1.2.44.
  - *Libglib2.0-dev* (archivos de desarrollo de la librería GLib). Versión actual 2.28.6.
  - *Xorg-dev* (librerías de desarrollo del sistema de ventanas de X.Org). Versión actual 7.6.
- **Rcssmonitor-classic**: alternativa al programa Rcssmonitor. En caso de encontrar incompatibilidades en el sistema que impidan su instalación, ésta versión proporciona un entorno visual totalmente compatible con las herramientas anteriores, aunque su desarrollo ha sido paralizado. La última versión disponible es la 11.1.0, y para instalarla se requiere la siguiente librería:
  - *Libstdc++6* (librería GNU estándar de desarrollo C++). Es necesario disponer además de la versión libstdc++5 para ejecutar algunos clientes anteriores al 2005.
- **Rcsslogplayer**: *Log Player*, herramienta para observar partidos pasados a través de los *logs* generados, es opcional. La versión utilizada es la 15.0.0, versión actual. Posee las mismas dependencias que el programa *Soccer Monitor*.

En su momento todos estos programas no eran compatibles con versiones del compilador GCC 4.5.2 o superior, así que se estudió la posibilidad de realizar las modificaciones oportunas en el código fuente de dichos programas. Por fortuna gracias a las listas de distribución de la sección de simulación de la RoboCup [48] se pudo llegar al equipo japonés Tokyo Tech [49], que se encontró

con el mismo problema a la hora de utilizarlo con el compilador GCC 4.0 y liberó unos parches que aplicados sobre el código fuente de los programas de la RoboCup solucionaron este problema.

Los programas enumerados han de ser instalados respetando el orden, puesto que todos ellos necesitan la instalación previa de *Rcssbase*. Una vez descargados todos los archivos con el código fuente y resueltas todas las dependencias, el procedimiento a seguir es el siguiente:

```
$ tar xzvf rcssbase-12.1.3.tar.gz
$ cd rcssbase-12.1.3
$ ./configure
$ make
$ sudo make install
$ cd ..
$ tar xzvf rcssserver-14.0.3.tar.gz
$ cd rcssserver-14.0.3
$ ./configure --prefix=/home/user/Downloads/rcssbase-12.1.3
$ make
$ sudo make install
$ cd ..
$ tar xzvf rcssmonitor-14.1.1.tar.gz
$ cd rcssmonitor-14.1.1
$ ./configure
$ make
$ sudo make install
$ cd ..
$ tar xzvf rcsslogplayer-14.0.1.tar.gz
$ cd rcsslogplayer-14.0.1
$ ./configure
$ make
$ sudo make install
$ cd ..
```

Comandos utilizados:

- *tar xzvf fichero.tar.gz* -> descomprime el fichero con extensión .tar.gz en un directorio con el mismo nombre.
- *./configure* -> configura el programa con las variables de entorno de la máquina. En el caso de la configuración del *Soccer Server* es necesario poner como prefijo la ruta de instalación del *Soccer Base*.
- *make* -> compila el programa
- *sudo make install* -> instala el programa con permisos de super usuario. Esta notación es propia distribuciones Debian, en otros linux el comando ha de ser en dos pasos, primero teclear *su* para obtener permisos y después *make install* para realizar la instalación.

Además se debe instalar el software de aprendizaje automático ***Weka***. Dicho software se puede descargar de su página web oficial [50]. Al descargarlo proporciona un archivo con extensión .zip que contiene el programa, que al estar basado en java es multiplataforma y puede ejecutarse tanto en Windows como en Linux.

Por último, a la hora de ejecutar los scripts de lanzamiento de las simulaciones, es necesario contar con la **shell CSH**, que está disponible para descarga automática desde los repositorios de Linux. Esta shell se utiliza para lanzar el script *Trainer/ATAQUE_AI2.sh*, que se explicará más adelante.

B.3 PROGRAMAS PROPIOS DEL PROYECTO

Una vez configurado el entorno que proporciona la RoboCup, el siguiente paso es confirmar que se disponen de todos los agentes necesarios para realizar las simulaciones y los programas requeridos para el proceso de generación del etiquetador y del modelador. A continuación se muestran los diferentes programas utilizados.

B.3.1 AGENTES DE LA ROBOCUP

Los diferentes agentes que participan en las ejecuciones son los siguientes:

- *Cliente CMUnited 99*: el delantero, cuyo objetivo es meter goles y observar el comportamiento del portero rival.
- *El portero rival*: el agente del que se desea modelar su comportamiento.
- *Trainer*: encargado de configurar las simulaciones, colocando a los jugadores, almacenando los resultados, etc.

B.3.2 PROGRAMAS PARA EXTRAER INFORMACIÓN DE LOS LOGS

En este apartado se enumeran las utilidades empleadas para realizar el procesamiento de los datos.

- *PorteroLog*: aplicación que extrae información útil del *log* generado por el servidor.
- *Ticker*: aplicación utilizada a la hora de crear el etiquetador, une el fichero resultante del *PorteroLog* con el *log* generado por el propio cliente CMUnited 99 y genera un archivo de datos con información por ejecuciones.
- *Ticker_predict*: programa utilizado a la hora de crear el modelador, a través del fichero de *log* generado por el propio cliente CMUnited 99 genera un archivo de datos con información por pares de ejecuciones.
- *MakeArff*: programa que genera ficheros legibles por el programa *Weka* a partir de la salida del programa *Ticker*. Para su funcionamiento requiere de un fichero de cabecera con la información de los diferentes atributos del conjunto de datos del etiquetador.
- *MakeArff_predict*: aplicación que genera ficheros legibles por el programa *Weka* a partir de la salida del programa *Ticker_predict*. Para su funcionamiento requiere de un fichero de

cabecera con la información de los diferentes atributos del conjunto de datos del modelador.

- *Rule2C*: aplicación que convierte los resultados del clasificador M5 en reglas comprensibles por el lenguaje C, para posteriormente ser incorporados al etiquetador o al modelador, dependiendo de en qué momento se utilice dicho programa.
- *Part2C*: programa que convierte los resultados del clasificador PART en reglas comprensibles por el lenguaje C, para posteriormente ser incorporados al etiquetador o al modelador, dependiendo de en qué momento se utilice dicho programa.
- *ParseReglas*: programa que corrige ciertos errores conocidos en la salida de los programas *Rule2C* y *Part2C*, puesto que el formato de reglas del *Weka* tiene sus peculiaridades, y no es completamente compatible con el estándar de C.

Varios de estos programas, concretamente *Ticker*, *Ticker_predict*, *MakeArff* y *MakeArff_predict*, han sido evolucionados de acuerdo al punto 3.2.4.7, dando como resultado las versiones numéricas de dichos programas. En el manual de usuario se hace referencia siempre a las versiones numéricas de dichos programas.

B.3.3 FICHEROS DE CONFIGURACIÓN EXTERNOS

En el presente punto se tratan los ficheros de configuración externos utilizables
Se ha creado un archivo externo llamado *config_params.conf* para la carga de parámetros a través de un fichero de configuración externo, con el objetivo de poder realizar cambios en los agentes sin necesidad de recompilar su código fuente.

Dicho archivo es utilizado tanto por el agente CMUnited 99 como por el *Trainer*, con lo que para no tener archivos duplicados y arriesgarse a una pérdida de consistencia se ha optado porque uno de ellos sea un enlace simbólico al otro. Bajo entorno linux dicho enlace se realiza con el comando *ln -s origen*.

En el caso de que se deseen externalizar más opciones en un futuro, ambos programas constan de un fichero de código llamado *utils.C* que contiene la función *LoadParameter*, que carga parámetros de configuración del archivo de configuración anteriormente descrito, con lo que únicamente habría que crear un string o cadena con el nombre del parámetro a cargar y pasárselo a la función. Es decir, el escalamiento de lectura externa de parámetros es prácticamente directo.

El fichero es el siguiente, con los siguientes parámetros:

- *FLAG_ETIQ*: modificación del agente CMUnited 99 para generar ficheros de *logs* utilizables para la construcción del etiquetador o del modelador.

- *USE_DEFAULT_SERVER*: reconocimiento del archivo de configuración del *Soccer Server*. La motivación de la utilización de este parámetro está explicada a continuación del archivo completo de configuración.
- *NUM_MOVEMENTS*: número de ejecuciones por cada iteración del *Trainer*.
- *KICK_MODEL*: tipo de modelo a utilizar por el cliente CMUnited 99.

El archivo completo de configuración se encuentra a continuación. En los comentarios se puede leer en qué fichero se utiliza cada parámetro, en qué afecta y cuál es la configuración por defecto:

```
# Configuracion del etiquetador/modelador/simulador
# Las lineas que empiezan con # son comentarios
# El formato es: parametro espacio valor

# Utilización del etiquetador o del modelador a la hora de generar los logs (para
CMU/MemPosition.C)
# Formato del modelador: [9 enteros] [18 dobles] acción [2 dobles]
# Formato del etiquetador: [9 enteros] [18 dobles] acción [2 dobles] acción [8
dobles]
# Opciones: modelador (0), etiquetador (1)
# (por defecto 1)
FLAG_ETIQ 0

# Reconocimiento o no del archivo de configuracion por defecto del servidor
rcssserver (para CMU/MemOption.C o support_CMU/shared/opt-code.C)
# La alternativa es tomar el server.conf del CMUnited 99 (acción 0)
# (por defecto 1)
USE_DEFAULT_SERVER 1

# Numero de jugadas de ataque del cliente CMUnited 99 en cada iteración del Trainer
(para trainer/TMmodel2.C)
# (por defecto 30)
NUM_MOVEMENTS 30

# Modelo a utilizar por el cliente CMUnited 99
# Opciones: sin modelo/kick_soft (0), con modelo/kick_soft_model (1),
aleatorio/kick_aleatorio (2), árbol de decisión/kick_arbol (3)
# (por defecto 0)
KICK_MODEL 1
```

En referencia al parámetro *USE_DEFAULT_SERVER*, en el estado inicial del proyecto tanto el delantero CMUnited 99 como el *Trainer* tomaban los parámetros de configuración inicial (como puede ser puerto de escucha del servidor, *host*, anchura de la portería, tamaño de la bola, energía inicial, porcentaje de decrecimiento de la energía, márgenes para golpear/parar la bola, etc) de unos ficheros de texto *server.conf* con un formato específico procedente del agente base vencedor de la competición del año 1999. Para no ceñirse únicamente a ese formato de ficheros, se ha modificado la lectura de dicho fichero para aceptar el formato estándar proporcionado por el *Soccer Server*. El servidor proporciona un fichero de configuración por defecto que se puede encontrar en *$HOME/.rcssserver/server.conf*. Para cargar este fichero únicamente hay que

modificar el parámetro *USE_DEFAULT_SERVER* del fichero *config_params.conf* (se carga automáticamente por defecto).

Muchos de los parámetros del nuevo *server.conf* no se utilizan en el estado actual del proyecto. Estos parámetros están definidos en el fichero de cabecera *CMU/MemOption.h* y se almacenan en el fichero de código fuente *CMU/MemOption.C*, para permitir su futura utilización si fuera necesaria.

B.4 CREACIÓN DEL ETIQUETADOR

Como se explicó en el punto 3.2.2, el proceso de creación del etiquetador consta de varios pasos, entre ellos la obtención de datos del comportamiento del agente rival basada en la observación, la fusión de esos datos con datos reales obtenidos a través del *Soccer Server*, y el tratamiento de estos datos para finalmente obtener unas reglas de comportamiento que se añadirán en el etiquetador. La Figura 23 muestra el esquema global de implementación del etiquetador con nombres de ficheros y programas reales:

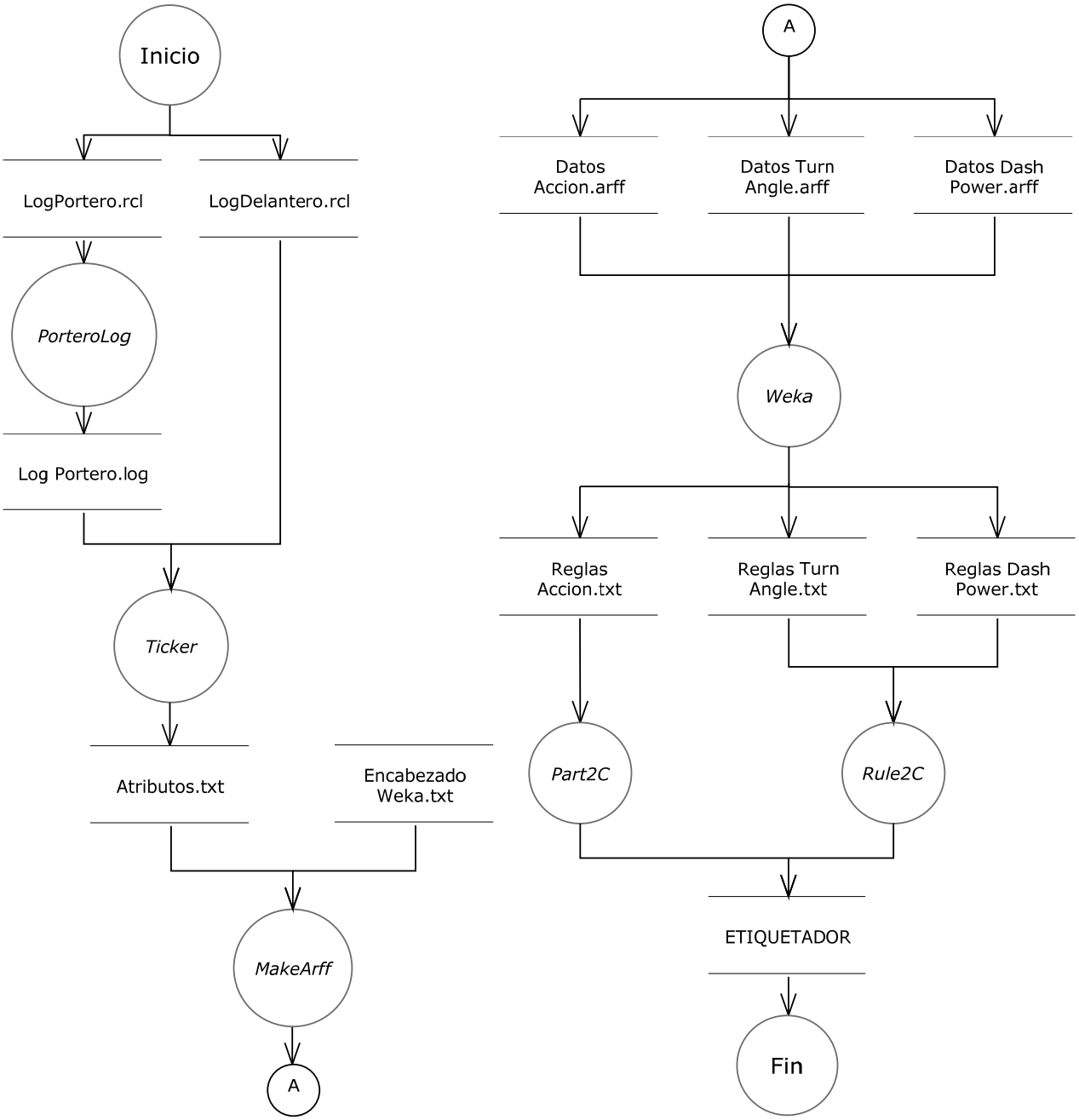


Figura 23: Esquema de implementación del etiquetador

Para utilizar el etiquetador hay que asegurarse de modificar un parámetro en archivo de configuración *config_params.conf* (se puede consultar el manual de referencia para ver las distintas posibilidades y comprender la utilidad de cada parámetro). Dicho parámetro es *FLAG_ETIQ*, el cual debe tener el valor de verdadero, 1. El resto de parámetros (salvo el número de jugadas por ejecución) toman valores deseados por el usuario, ya que no influyen directamente a la hora de generar el etiquetador.

Los puntos a seguir para generar el etiquetador de acciones son los siguientes:

- **Obtención de datos**: el primer paso consiste en tomar datos del agente a observar. Para ello se ejecuta dos veces una simulación de cincuenta jugadas. Para ello hay que modificar el parámetro *NUM_MOVEMENTS* en el archivo *config_params.conf* poniendo el valor de 50 a este parámetro.
- **Tratamiento del fichero de *log* generado por el servidor**: para extraer la información útil de este fichero, se utiliza la herramienta *PorteroLog*.
- **Fusión de los datos reales con los datos percibidos**: este proceso se realiza mediante la herramienta *Ticker*.
- **División por componentes de los datos**: a este fichero procesado se le trata con el programa *MakeArff*, que genera una serie de archivos de datos que tomará la herramienta de minería de datos *Weka*. El programa *MakeArff* además toma un fichero de texto como parámetro con la información de cabecera necesaria para que el archivo sea comprensible por *Weka*.
- **Inclusión de los ficheros de reglas en el etiquetador**: Sobre el resultado de utilizar el clasificador correspondiente sobre el fichero de clases, se aplica el programa *Part2C*, que genera un documento con reglas comprensibles por el lenguaje de programación C.
  A su vez, sobre el resultado de utilizar el clasificador correspondiente sobre el fichero de cada uno de los tipos particulares *turn* y *dash*, se aplica el programa *Rule2C*, que también genera un documento con reglas comprensibles para el agente.
  Sendos ficheros de reglas generados se incorporan a mano al fichero *Reglas.C*, pasando antes por el programa *parseReglas* para solucionar un defecto conocido.

A continuación se detalla cada uno de estos pasos en profundidad.

B.4.1 OBTENCIÓN DE DATOS

La forma de obtener los resultados es la siguiente: simplemente se procede a simular un gran contenido de partidos en rondas de 30 partidos cada uno.

Para ello se utilizan los scripts de ejecución controlada que se pueden encontrar en el directorio del *Trainer*. Esos fichero son el script *E100_AICOM2.sh*, que lanza las n ejecuciones deseadas, y

el script *ATAQUE_AI2.sh*, que contiene las llamadas a los agentes que se van a lanzar. Ambos ficheros son explicados en profundidad en el Anexo C.

El primer paso que debemos dar para realizar la ejecución inicial es modificar el fichero de configuración *config_params.conf* configurando el parámetro *FLAG_ETIQ* para que el *log* del agente sea en formato apropiado para el etiquetador, con lo que hay que asegurarse que el valor de *FLAG_ETIQ* sea igual a 1, es decir, verdadero.

Los pasos a realizar para hacer una simulación de partido son los siguientes:

1. Puesta en marcha del **servidor**: se realiza a través del comando *"rcssserver".*
2. Puesta en marcha del **monitor**: a través del comando *"rcssmonitor".*
3. Ejecución del **delantero**: se ha de utilizar el comando "*CMU/charliclient -file server.conf -file client.conf"* para su ejecución. Las opciones de entrada son ficheros de configuración que se cargan con el cliente.
4. Ejecución del **portero** rival: al tratarse con varios porteros diferentes cada uno tiene su propio comando de inicialización, en la sección de pruebas de este proyecto se pueden ver las diferentes llamadas.
5. Ejecución del ***Trainer***: para ejecutar se ha de lanzar por línea de comandos lo siguiente: "*support_CMU/trainer/trainer -file server.conf -file trainer.conf -epochs_to_run 1 -epoch_start 0*". Los diferentes parámetros de entrada se refieren a los ficheros de configuración a cargar y a las ejecuciones que se van a lanzar.

Para facilitar las llamadas se puede utilizar el script *trainer/ATAQUE_AI2.sh*, que ejecuta secuencialmente cada uno de los anteriores programas y almacena la salida de cada elemento en un fichero de *log*, que posteriormente puede ser utilizado para comprobar el correcto funcionamiento de cada uno de los programas.

En la Figura 24 se ve la ejecución por consola del *Trainer*, en la que se ve el número de ejecución, los mensajes que mandan cada uno de los clientes al servidor, y las jugadas realizadas (si es gol, parada del portero, o si el balón salió fuera, en el caso de la captura de pantalla es un balón parado por el portero, con la consiguiente acción *save*):

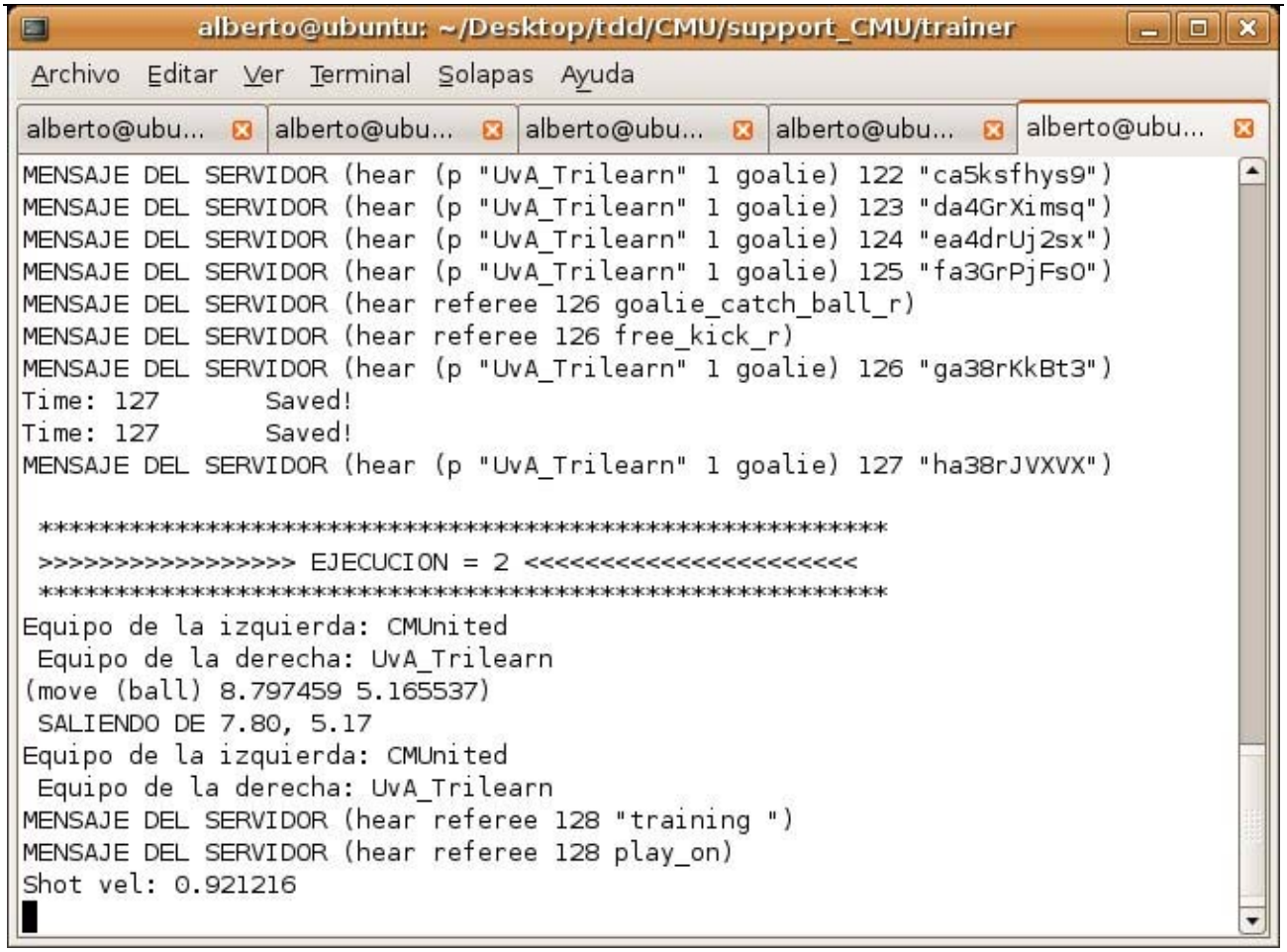


Figura 24: Ejecución del agente Trainer

En la Figura 25 se muestra una captura de pantalla de un instante del partido, con los equipos CMUnited 99 (delantero) y UvA Trilearn 2003 (portero), siempre a través del visor de sucesos *Rcssmonitor*, que aporta una interfaz gráfica para seguir en tiempo real las evoluciones de los jugadores:

Figura 25: Campo de juego visto a través del RoboCup Soccer Monitor

Esta ejecución genera dos tipos de registros, los obtenidos por el delantero a través de la observación, y los obtenidos por el servidor, estos últimos datos verídicos totalmente objetivos:

1. **Logs obtenidos del propio delantero del CMUnited 99**, disponibles en el directorio CMU/Logfiles:
   - *CMUnited1-l.log*: fichero que contiene toda la información que se envía al servidor.
   - *CMUnited1-l-actions.log*: registro por parte del delantero de todos los parámetros observados en cada instante, como la posición actual del agente, la energía disponible, la ubicación de la bola, y las acciones que se llevan a cabo en cada momento por parte del delantero.
   - *CMUnited1-l-kickM.log*: registro con un histórico de los instantes en los que se ha ejecutado la acción de chutar a puerta.
   - *CMUnited1-l-oppnt.log*: fichero que contiene la información del comportamiento del portero que ha sido observada por el delantero, éste fichero será utilizado más adelante para generar el etiquetador.

2. **Logs generados por el propio servidor**, disponibles en el directorio desde donde se ha ejecutado el servidor:

   - *fecha-CMUnited_resultado-vs-equipo_rival.rcg*: archivo binario que genera el servidor, contiene toda la información del partido jugado. Este tipo de archivo puede reproducirse con la herramienta *rcsslogplayer*.
   - *fecha-CMUnited_golesCMUnited-vs-equipo_rival_golesEquipoRival.rcl*: registro de todos los mensajes que se mandan al servidor, será utilizado más adelante para cotejar las acciones que el delantero cree que el portero está realizando con las acciones que realmente está realizando. Un ejemplo de fichero con extensión *.rcl* es *201201011200-CMUnited_5-vs-UvA_Trilearn_0.rcl*.

Una vez obtenidos los datos, el siguiente paso es tratarlos para poder utilizarlos en el futuro. El fichero *CMUnited1-l-oppnt.log* tiene información concreta que el agente ha percibido, información válida que se utilizará más adelante, pero el fichero de *log* generado por el servidor, con extensión *.rcl*, no es útil en el estado en el que se emite, por lo cual se precisa de la utilización de herramientas intermedias, suceso que se explica en el siguiente punto.

B.4.2 MODIFICACIÓN DEL LOG GENERADO POR EL SERVIDOR

En este apartado se van a explicar una serie de modificaciones necesarias que hay que aplicar al fichero de *log* generado por el servidor para obtener información que sea relevante y pueda ayudar en el proceso de generación del etiquetador.

El primera paso que hay que dar es eliminar la información correspondiente al delantero y al servidor, y unificar el formato con el objetivo de conseguir información únicamente de las acciones del delantero rival.

Para llevar a cabo dicho proceso se utiliza la herramienta *Porterolog*, a la que se le pasa el archivo *.rcl* generado por el servidor y devuelve un fichero con el mismo nombre y extensión *.log*:

- *porterolog log_server.rcl*
  - *porterolog 201201011200-CMUnited_5-vs-UvA_Trilearn_0.rcl*

Ejemplo de fichero .rcl:

```
0 Recv UvA_Trilearn_1: (attentionto off)(turn_neck 0)
0 (referee kick_off_l)
0 Recv Coach: (ear on)
0 Recv Coach: (init (version 7))
0 Recv Coach: (eye on)
1 Recv Coach: (say training)
1 (referee training)
1 Recv Coach: (move (player UvA_Trilearn 1) 49.500000 0.000000 34.748749)
1 Recv Coach: (move (ball) 10.390972 -12.907817)
1 Recv Coach: (move (player CMUnited 1) 9.390972 -12.907817 0.000000)
1 Recv Coach: (change_mode play_on)
```

```
1  (referee play_on)
1  Recv UvA_Trilearn_1: (attentionto off)(turn_neck 0)
2  Recv UvA_Trilearn_1: (dash 100)(attentionto off)(turn_neck 0)
3  Recv UvA_Trilearn_1: (turn 180)(attentionto our 9)(turn_neck 65)
4  Recv UvA_Trilearn_1: (turn 128)(attentionto our 9)(turn_neck -83)
5  Recv UvA_Trilearn_1: (dash 100)(attentionto our 9)(turn_neck 0)
6  Recv UvA_Trilearn_1: (turn 92)(turn_neck -37)
7  Recv CMUnited_1: (kick 28.03 -73.29)
7  Recv CMUnited_1: (turn_neck -85.69)
7  Recv UvA_Trilearn_1: (turn 28)(turn_neck -18)
8  Recv CMUnited_1: (turn -79.22)
8  Recv UvA_Trilearn_1: (turn -1)(turn_neck 1)
9  Recv UvA_Trilearn_1: (turn -4)(turn_neck 2)
9  Recv CMUnited_1: (turn 52.22)
10 Recv CMUnited_1: (turn -36.34)
10 Recv UvA_Trilearn_1: (dash 100)(turn_neck 0)
11 Recv CMUnited_1: (dash 100.00)
11 Recv UvA_Trilearn_1: (turn -70)(turn_neck 32)
12 Recv UvA_Trilearn_1: (dash 100)(turn_neck 1)
12 Recv CMUnited_1: (dash 100.00)
13 Recv CMUnited_1: (dash 100.00)
13 Recv UvA_Trilearn_1: (turn 94)(turn_neck -41)
14 Recv CMUnited_1: (dash 100.00)
14 Recv UvA_Trilearn_1: (turn -10)(turn_neck 6)
15 Recv UvA_Trilearn_1: (dash 59)(turn_neck 3)
15 Recv CMUnited_1: (kick 61.69 113.06)
15 Recv CMUnited_1: (turn_neck 144.69)
16 Recv CMUnited_1: (turn 144.15)
16 Recv UvA_Trilearn_1: (turn_neck 0)
17 Recv CMUnited_1: (dash 100.00)
17 Recv UvA_Trilearn_1: (turn_neck 0)
18 Recv UvA_Trilearn_1: (dash -62)(turn_neck 0)
```

Se ve que a la izquierda sale el momento en segundos en que se recibió el mensaje por parte del servidor, y a la derecha el mensaje recibido, precedido por el nombre del cliente que envió el mensaje, que puede ser el delantero (líneas con el texto "*CMUnited*"), el portero (en este caso con el mensaje "*UvA_Trilearn*"), el *Trainer* (mensajes con "*coach*") o mensajes enviados por el propio servidor (como "*kickoff*", para iniciar la jugada). Una vez ejecutado el programa *porterolog*, lo que se obtiene es la siguiente trama:

```
0, 0, turn, 179, 0
2, 0, dash, 100, 0
3, 0, turn, 180, 0
4, 0, turn, 128, 0
5, 0, dash, 100, 0
6, 0, turn, 92, 0
7, 0, turn, 28, 0
8, 0, turn, -1, 0
9, 0, turn, -4, 0
10, 0, dash, 100, 0
11, 0, turn, -70, 0
12, 0, dash, 100, 0
13, 0, turn, 94, 0
14, 0, turn, -10, 0
15, 0, dash, 59, 0
18, 0, dash, -62, 0
```

El resultado final es por tanto un listado de las acciones que ejecuta el portero, con los parámetros de dichas acciones (fuerza de avance, ángulo de giro) y el instante preciso en que se ejecuta. Por lo que se puede comprobar, se han dejado únicamente los siguientes datos:

- El instante en que se envía el mensaje
- El número de iteraciones en el mismo instante (un agente puede mandar varios mensajes al servidor en el mismo instante de tiempo)
- La acción que realiza el delantero (*turn*, *dash*, *catch*, *kick*, *none*)
- Los parámetros de dicha acción (fuerza y/o ángulo)

Así pues tomando la última línea se obtienen los siguientes datos:

```
18, 0, dash, -62, 0
```

En el instante de tiempo *t=18* se produjo la acción "avanzar", con una fuerza de -62 (con fuerzas negativas se retrocede). El segundo parámetro, es decir el primer 0, indica que no hubo más acciones en ese instante de tiempo. El último 0 indica que la acción dash no toma más parámetros.

B.4.3 FUSIÓN DE LOS DATOS DE AMBOS AGENTES

Una vez obtenido el fichero con extensión *.log*, el siguiente paso es utilizar la herramienta *ticker*, cuya función radica en cotejar los datos de ambos agentes, tanto del delantero que contiene la información percibida, como del portero, que son los datos reales obtenidos a través del servidor, y plasmarlos en un archivo que más tarde será utilizado para obtener los datos necesarios para el programa *Weka*.

Para hacer uso de la herramienta anterior, se le pasan como argumentos los logs que hemos obtenido en los pasos anteriores, tanto el *log* generado por el propio cliente, como el *log* generado por el servidor y que ha sido posteriormente tratado con la herramienta *Porterolog*:

- *ticker -i log_oponente_CMU -c fichero_clases_porterolog -v tamaño_ventana -o salida*
    - *./ticker -i CMUnited1-l-oppnt.log -c* 201201011200-*CMUnited_5-vs-UvA_Trilearn_0.log -v 2 -o fichero_salida_ticker.log*

El tamaño de ventana consiste en el número de cadenas que va a tomar a la vez para sacar los datos. Se utiliza un tamaño por defecto de 2, para poder contar de dos en dos instantes de tiempo.

Debido a la opción escogida para etiquetar, si se observa el fichero *log_oponente_CMU*, es decir el fichero con la información observada por el delantero, se observa que tiene el siguiente formato: *[9 enteros] [18 dobles] acción [2 dobles] acción [8 dobles]*, debido a que almacena la información percibida por el delantero, la acción que toma el mismo delantero, y la acción que piensa que

toma el portero. Es reseñable porque a la hora de modelar el fichero tendrá un formato distinto, y además las herramientas utilizadas producirían ficheros vacíos en caso de recibir como entrada un fichero con tramas diferentes a las esperadas.

El fichero de salida tiene el siguiente formato (se han tomado seis tramas al azar):

```
63,0,1,1,0,1,1,1,0,49.36,-2.03,0.99,13.23,0.03,-23.71,8.29,38.3,-
4.98,0.95,1.55,1.82,-33.92,-1.92,36.49,-5.07,37,0.15,kick,-
55.44,100,62,0,0,1,0,1,1,1,1,49.04,-2.05,0.98,13.14,0.01,-23.12,3.88,36.82,-
4.66,0.9,0.42,0.76,8.65,35.65,36.29,-5.21,37,0.37,dash,0,100,desconocida,0,0

64,0,1,1,0,1,1,1,0,49.46,-2.1,0.99,13.23,0.05,-21.38,10.62,41.52,-
5.64,0.95,2.21,4.93,-39.74,-7.74,36.61,-5.24,35.08,0.06,turn,-
3.36,0,63,0,1,1,0,1,1,1,0,49.36,-2.03,0.99,13.23,0.03,-23.71,8.29,38.3,-
4.98,0.95,1.55,1.82,-33.92,-1.92,36.49,-5.07,37,0.15,kick,-55.44,100,dash,100,0

65,0,0,1,0,1,1,1,0,49.41,-2.12,0.98,13.12,0.02,-13.77,18.23,43.72,-
5.93,0.9,2.08,7.1,-33.23,-1.23,36.66,-5.2,27.34,0.03,turn,-
10.06,0,64,0,1,1,0,1,1,1,0,49.46,-2.1,0.99,13.23,0.05,-21.38,10.62,41.52,-
5.64,0.95,2.21,4.93,-39.74,-7.74,36.61,-5.24,35.08,0.06,turn,-3.36,0,dash,100,0

69,0,1,1,1,1,1,0,0,48.77,-6.12,0.99,12.2,0,-30.86,1.14,48.75,-6.34,0.95,0,12.2,-
31.86,0.14,36.62,-5.06,25.86,0,turn,-0.14,0,68,0,0,1,0,1,1,1,0,48.84,-
5.13,0.98,12.21,0.06,-26.12,5.88,50.81,-6.64,0.9,1.95,14.28,-32.14,-0.14,36.63,-
5.05,25.75,0,turn,-0.25,0,turn,-44,0

71,0,0,1,0,1,1,1,0,47.8,0.64,0.98,40.4,0,-18.14,-
18.14,12.21,16.94,0.9,0.33,1.33,0.13,0.13,10.89,17.08,-5.86,0,kick,-
16.45,29.39,70,0,1,1,0,1,1,0,1,47.8,0.64,0.99,40.4,0,-
18.14,13.86,11.89,17.08,0.95,0,1,5.86,37.86,10.89,17.08,-5.86,0,turn,-
31.86,0,kick,17,-35

72,0,1,1,0,1,1,1,0,47.8,0.64,0.99,39.84,0,-20.3,-20.3,13.03,16.7,0.95,0.55,1.58,-
8.39,-8.39,11.49,17.03,-
4,0.25,dash,0,100,71,0,0,1,0,1,1,1,0,47.8,0.64,0.98,40.4,0,-18.14,-
18.14,12.21,16.94,0.9,0.33,1.33,0.13,0.13,10.89,17.08,-5.86,0,kick,-
16.45,29.39,kick,17,-36
```

Si consideramos que la acción se lleva a cabo en el instante *t*, la información que estamos obteniendo es la concatenación de los datos percibidos en el instante *t* y en el instante *t-1* además de la acción concreta realizada en dicho instante y los parámetros con que se ejecutó.

Cabe destacar que en esta trama de datos y en las tramas sucesivas se van a representar las acciones que realizan tanto el delantero como el portero de manera textual, es decir, que se verá "*kick*", "*dash*" o la acción determinada. Sin embargo, tanto a la hora de generar el *log* del delantero como en el tratamiento que se da a los datos en los ficheros intermedios, como pueden ser *Ticker*, *MakeArff*, y otros programas, esta información se trata de manera numérica, utilizando la conversión mostrada en la Tabla 43:

TIPO DE ACCIÓN	REPRESENTACIÓN
None	0
Dash	1

TIPO DE ACCIÓN	REPRESENTACIÓN
Turn	2
Kick	3
Catch	4
No_hay	98
Descono	99

Tabla 43: Representación numérica de las diferentes acciones posibles

B.4.4 GENERACIÓN DE LOS ARCHIVOS PROPIOS DE WEKA

Una vez obtenido el fichero anterior, el siguiente paso es generar los archivos preparados para ser utilizados con la herramienta de minería de datos *Weka*. Para ello se ejecuta el programa *MakeArff*. Esta herramienta precisa de un fichero de encabezado particular que contiene la información de los diferentes atributos que se han generado y permite a *Weka* interpretar directamente los datos.

La llamada al *MakeArff* es la siguiente:

- *makeArff -i fichero_generado_ticker -e encabezado*
    - *./makeArff -i fichero_ salida_ticker.log -e encabezado.txt*

El encabezado tiene el siguiente formato:

```
@relation pricipalDelanteroPrediccion

@attribute server     real
@attribute interno    real
@attribute veo        {1,0}
@attribute num        real
@attribute BallKickableForOpponent {1,0}
@attribute CanFaceOpponentWithNeck {1,0}
@attribute CanSeeOpponentWithNeck  {1,0}
@attribute BallMoving              {1,0}
...
@attribute Dif_MyBA                real
@attribute Dif_MyS                 real
@attribute CLASE                   {turn, kick, dash, catch, none, desconocida}

@data
```

En este archivo de encabezado se encuentran los parámetros que en su momento percibió el agente, ya explicados anteriormente, y por su parte el programa *Ticker* ha añadido los atributos calculados, es decir, las diferencias entre dos instantes de tiempo y las distancias euclídeas recorridas.

Al ejecutarse, se generan cinco archivos diferentes, todos ellos listos para ser ejecutados con *Weka*, por lo que requieren de la extensión *.arff*. Son los siguientes:

- *fichero_salida_**principal.arff***: archivo que contiene toda la información extraída, este fichero será utilizado para clasificar la acción que realiza el agente rival a través de la

información de la que se dispone en ese momento concreto, consta de unas 4000-5000 instancias. Así pues este conjunto de datos se utilizará para crear un clasificador que, a partir de los diferentes atributos disponibles, calcule la acción que va a llevar a cabo el portero.

- *fichero_salida_**turn.arff***: archivo que contiene la información referente a la clase *turn*, por lo cual este fichero contendrá la información de la que dispone el delantero cuando la acción del portero es la de girar, por lo que el clasificador determinará el ángulo de giro del portero rival cuando éste decide girar. Un fichero de este tipo estará compuesto de unas 1000-3000 instancias, dependiendo del portero a estudiar.
- *fichero_salida_**dash.arff***: archivo que contiene la información referente a la clase *dash*, dispone de la información de la que dispone el delantero cuando la acción del portero es la de avanzar, por lo que el clasificador determinará la fuerza de avance del portero rival cuando éste toma la acción de avanzar. Al igual que el fichero *turn*, suele estar formado por unas 1000-3000 instancias, dependiendo del portero a estudiar.
- *fichero_salida_**kick_angle**:* archivo que contiene la información referente a la clase *kick*, dispone de la información de la que dispone el delantero cuando la acción del portero es la de golpear la pelota, determinando el clasificador el ángulo del portero rival cuando éste toma dicha acción. El número de instancias *kick angle* suele ser reducido o inexistente, dependiendo del portero a estudiar.
- *fichero_salida_**kick_power**:* archivo que contiene la información referente a la clase *kick*, dispone de la información de la que dispone el delantero cuando la acción del portero es la de golpear la pelota, determinando el clasificador la fuerza de golpeo del portero rival cuando éste toma dicha acción. El número de instancias *kick power* es exactamente igual al número de instancias *kick angle*, puesto que ambos ficheros se generan partiendo del mismo origen de datos.

Las instancias generadas con la fuerza y el ángulo de golpeo de la pelota por parte del portero (*kick_power* y *kick_angle* respectivamente) han sido desarrolladas, pero no es habitual que surjan este tipo de instancias debido a que la instrucción que utilizan los porteros en el 99% de los casos cuando tienen acceso a la pelota es *catch* para capturarla.

Independientemente de la cantidad de instancias de este tipo de acciones, en caso de existir alguna el programa *MakeArff* generará estos ficheros de tipo *kick angle* y *kick power*. El tratamiento de este tipo de instancias es completamente análogo al de los ficheros de tipo *turn* o de tipo *dash*.

Estas acciones se corresponden con los comandos de control descritos en el punto 2.5.2 Funcionamiento del Soccer Server.

B.4.5 PROCESAMIENTO EN WEKA

El siguiente paso es emplear la herramienta de minería de datos *Weka*, que se utilizará para extraer información de los datos conseguidos.

Es necesario ejecutar esta herramienta pasando como parámetro un tamaño de memoria virtual asociada mayor al que viene por defecto, porque al trabajar con grandes cantidades de datos *Weka* se queda sin memoria con rapidez, puesto que la configuración por defecto utiliza 128MB de memoria para sus operaciones:

La llamada para su ejecución es la siguiente:

- *java -Xmx512m -jar weka.jar*

El parámetro 512m se refiere al número de MB de memoria RAM asociados al programa. Como ya se ha expuesto la cantidad por defecto es mucho menor y *Weka* se colapsa con facilidad a la hora de crear los clasificadores ante tal cantidad de datos que tiene que gestionar.

Al iniciar la aplicación se muestra una pantalla donde se permite escoger entre diferentes opciones, entre ellas *Explorer*, *Experimenter*, *Knowledge Flow* y *Simple CLI*. Para el uso que se le va a dar en este proyecto valen tanto la aplicación *Explorer* como la aplicación *Experimenter*. En las siguientes líneas se hace uso del modo *Explorer* porque está orientado a una aplicación más directa y facilita mejor la comprensión, puesto que el modo *Experimenter* está diseñado para realizar tareas secuenciales.

El primer paso consiste en comprobar que el fichero .*arff* se ha generado correctamente. Para ello basta con abrirlo desde el programa. Si la generación ha sido correcta se verá una pantalla como la que muestra la Figura 26:

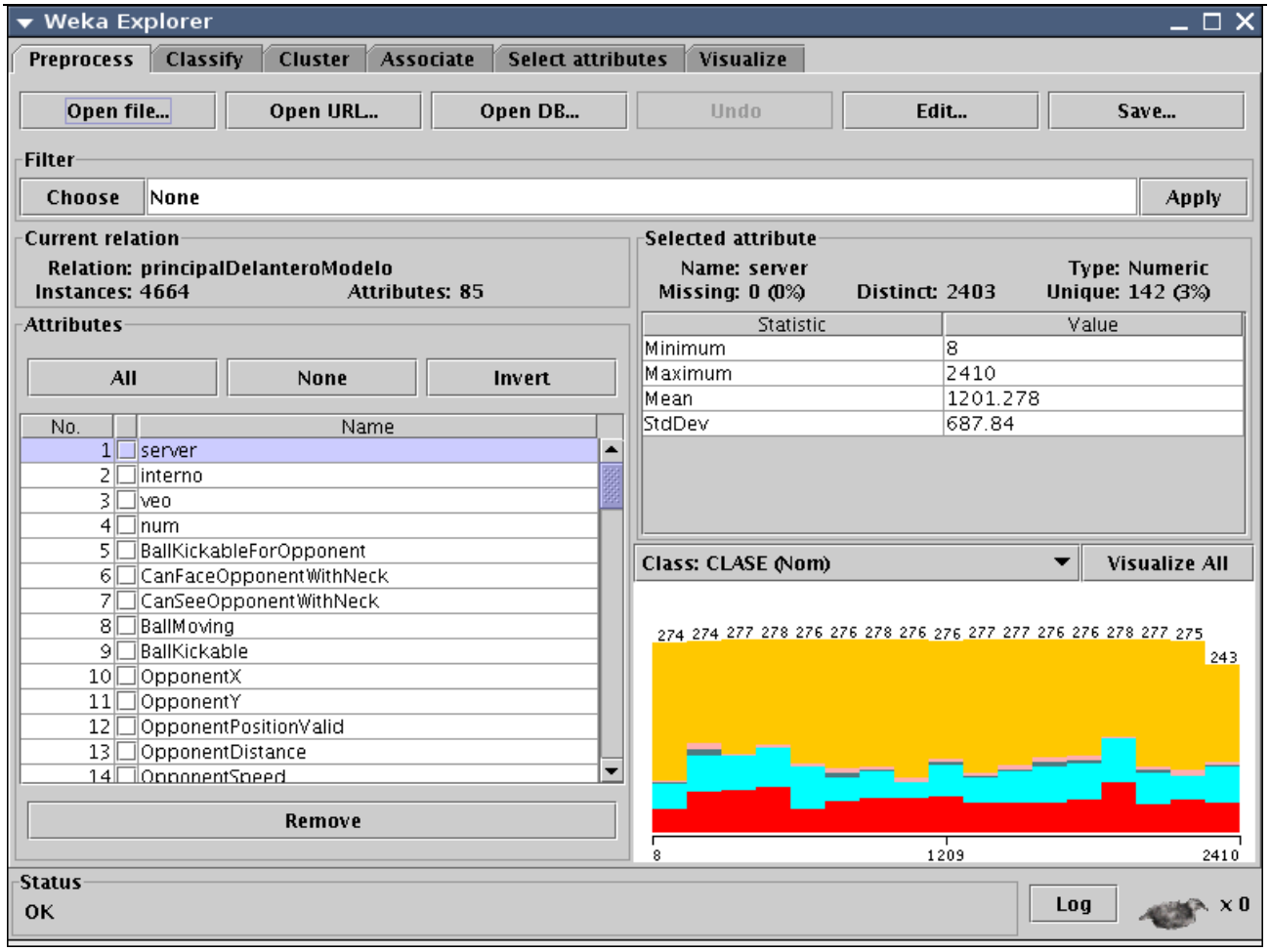


Figura 26: Weka explorer: carga del fichero *principal.arff*

El funcionamiento de esta aplicación es el siguiente: al programa *Weka* se le carga una serie de registros o instancias, todas ellas con la misma estructura, que consisten en una serie de pares atributo/valor. Uno de estos atributos representa la categoría de los registros, y precisamente los clasificadores se encargan de predecir la categoría de las instancias a través de una serie de reglas aplicadas a los atributos que no son dicha categoría.

Así pues el programa enseña el conjunto de datos disponible (en este caso se dispone de 4664 instancias), mostrando además los diferentes atributos de los que se componen los datos generados, así como una visualización en la zona inferior derecha de las instancias por atributo que corresponden a cada clase (ordenadas por colores para facilitar la comprensión).

La intención del procesamiento es establecer con la mayor certeza posible la clase de una instancia determinada a partir de los valores de sus atributos. Para ello se utilizan los algoritmos de clasificación. Previamente hay que eliminar ciertos atributos que no tienen información relevante para la clasificación, son "*interno*", "*num*" y "*server*", los dos primeros siempre van a tener el mismo valor y el tercero se refiere al instante concreto de tiempo, con lo que deduciría las acciones en función del tiempo de ejecución, lo cual es erróneo porque ningún partido es igual a otro debido a que hay un componente de aleatoriedad importante.

La forma de realizar esta purga es mediante un filtro específico, concretamente el filtro no supervisado de atributos llamado *Remove*. Así pues se ha de ejecutar el siguiente filtro: *weka.filters.unsupervised.attribute.Remove-R1-2,4,31-32,34*. Si se ha hecho correctamente se verá que el número de atributos desciende de 85 a 79.

Una vez preparado el fichero, se procede a realizar la clasificación en función de la clase. Como se ha explicado anteriormente, hay dos tipos de ficheros, el fichero *principal* y el fichero con los datos de *turn*, *dash*, *kick power* y *kick angle*, si existieran estos dos últimos. Estos ficheros son de diferentes tipos y tienen que ser tratados de forma diferenciada, como se muestra a continuación.

B.4.5.1 FICHERO .ARFF DE TIPO PRINCIPAL

Este punto tiene como objetivo presentar el fichero llamado *fichero_salida_principal.arff*, generado por el programa *MakeArff*, explicando qué datos contiene y qué se hace con esos datos.

Se llama principal porque contiene toda la información generada por el *ticker*, con lo que no se prescinde de ningún tipo de datos. La intención es clasificar todos los datos de los que se disponen para poder obtener la clase real que se ejecuta a través de esos datos, es decir, tomar todos los parámetros percibidos por el delantero, a los que se han añadido los atributos calculados de forma manual (a través del *Ticker*), y con esos datos construir un clasificador que diga cuál es la acción que realiza el portero. Ésa es la intención que se persigue al construir este clasificador, contar con una serie de reglas que sirvan para predecir la acción a realizar por el portero. El cómo es posible poder realizar la predicción se debe a que se cuenta con la acción real realizada, puesto que esta información ha sido proporcionada por el servidor. En cambio, a la hora de generar el modelador no se podrá contar con esta información, puesto que las decisiones del delantero han de ser tomadas en tiempo real en base únicamente a la información que pueda observar en ese momento y al modelo de conocimiento que habrá sido incorporado en él.

Al cargar el fichero con los datos de la acción, una vez eliminados los atributos superfluos, al seleccionar el atributo clase se muestra la pantalla visible en la Figura 27:

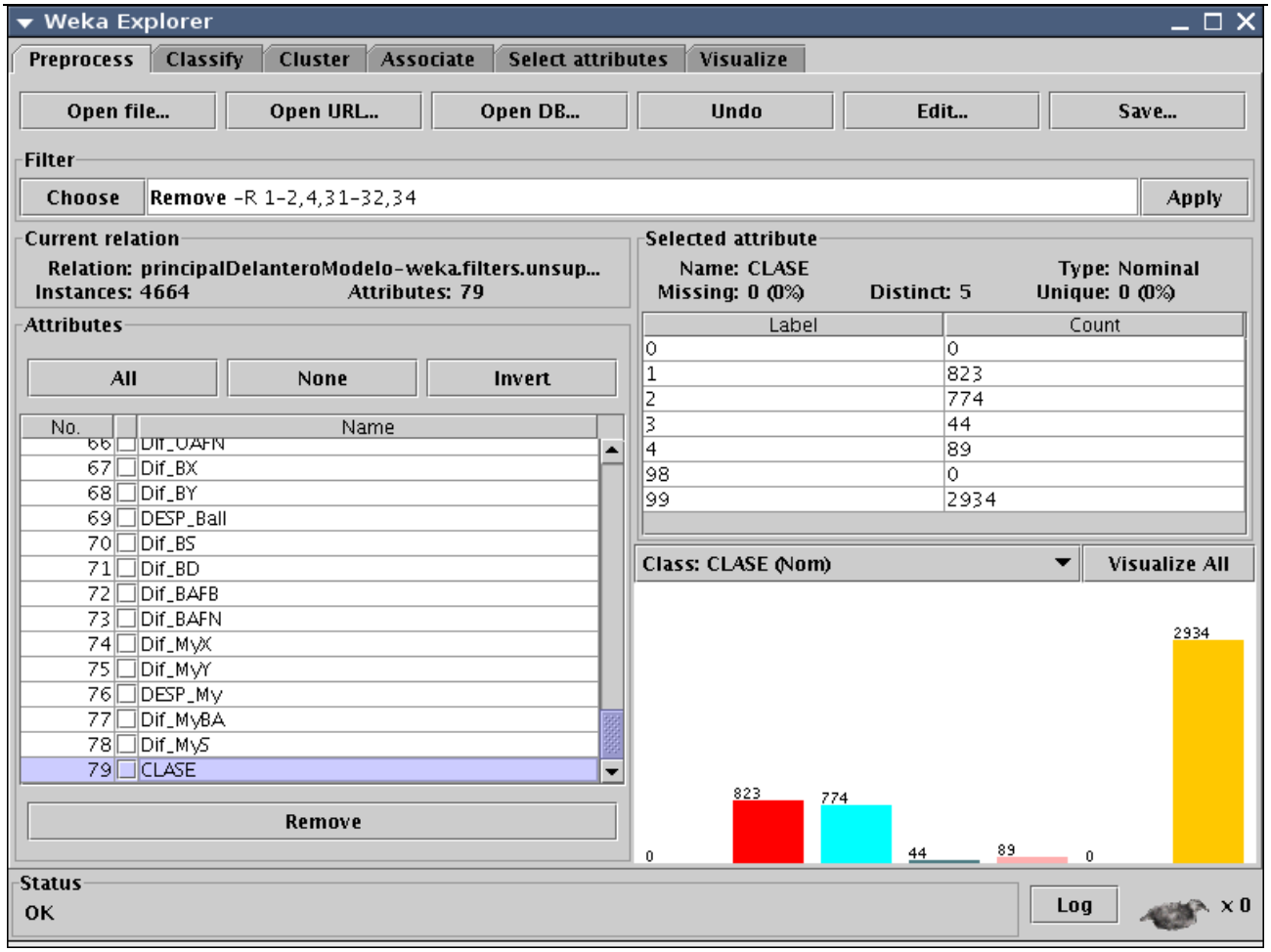


Figura 27: Weka explorer: clase del fichero de tipo principal

En ella se puede ver la relación creada, principalDelanteroPrediccion, sus atributos e instancias. Las clases posibles se ven en el último atributo, llamado CLASE, y son las siguientes:

- *None:* el jugador rival no realiza ninguna acción, equivale a la clase numérica 0
- *Dash*: avanza con la pelota, equivale a la clase numérica 1
- *Turn*: representa que el jugador gira, equivale a la clase numérica 2
- *Kick*: el jugador golpea la pelota, equivale a la clase numérica 3
- *Catch*: toma la pelota, equivale a la clase numérica 4
- *No_hay:* clase específica utilizada en el modelador, utilizada cuando el etiquetador que tiene incorporado no es capaz de predecir la clase, equivale a la clase numérica 98
- *Desconocida*: no se ha podido establecer, equivale a la clase numérica 99

Para clasificar este archivo *.arff* con todos los datos, como la clase es de tipo nominal ("*dash*", "*turn*", "*kick*", etc, aunque en realidad a la hora de realizar los ficheros de entrada del *Weka* se ha conservado el tipo nominal pero se utilizan la representación numérica) se ha escogido el **clasificador PART**.

El clasificador PART es la implementación en *Weka* del algoritmo de clasificador C4.5. Este tipo de algoritmo genera un árbol de decisión en el que cada nodo del árbol es un atributo de las diferentes instancias con que se trabajan, y cada rama representa un posible valor de ese atributo, siendo cada hoja la clase que se obtiene cuando se evalúan las diferentes condiciones a través de las que se ha pasado. En concreto el algoritmo C4.5 selecciona en cada nodo el atributo con mayor ratio de ganancia de información, mejorando otros árboles de decisión en que se basa, como puede ser el algoritmo ID3. Para más información sobre el tema de aprendizaje automático se recomienda consultar el trabajo de Alpaydin [17].

Se ha utilizado este algoritmo porque el árbol de reglas que genera es fácilmente convertible a formato C, ya que este paquete de reglas junto con los obtenidos en la clasificación de los ficheros de tipo *dash*, de tipo *turn*, de tipo *kick power*, y de tipo *kick angle* compondrán el etiquetador que será incorporado al delantero.

La siguiente imagen se muestra la llamada con las opciones por defecto más la opción *ReducedErrorPruning*, esta opción es utilizada para aumentar la sencillez del árbol de reglas generado, puesto que prueba las diferentes opciones de poda del árbol para hacerlo lo más sencillo posible sin empeorar el rendimiento. En la
Figura 28 se observa una captura del programa *Weka* con los datos ya cargados, donde se escoge el clasificador con las opciones requeridas.

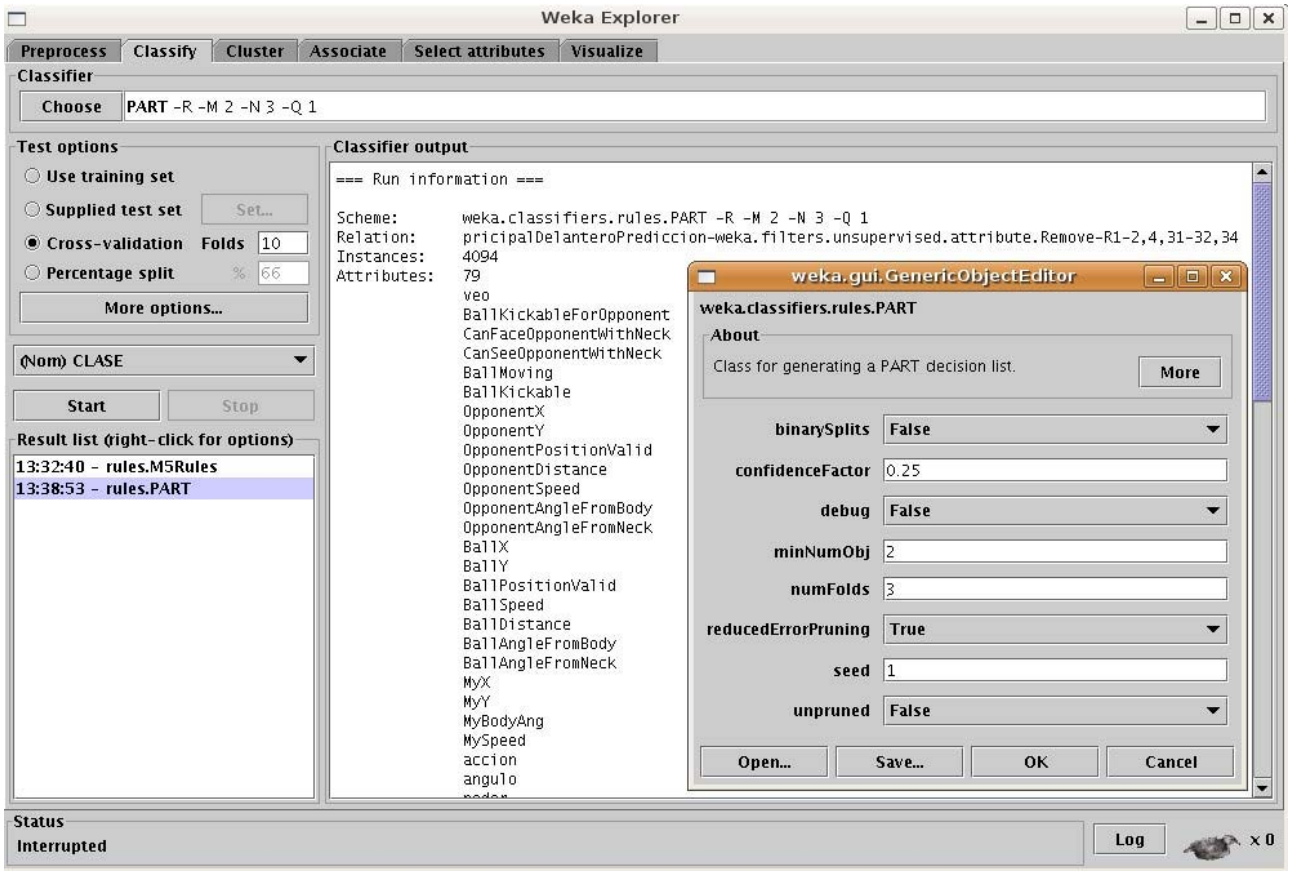


Figura 28: Weka explorer: opciones del clasificador PART

Como se puede ver, a la hora de realizar la clasificación se utiliza la opción de validación cruzada con diez contenedores, puesto que en principio no se puede aportar un conjunto de entrenamiento contra el que clasificarse, ya que son todos los datos de los que se disponen.

Los resultados obtenidos (en este caso corresponden a un conjunto de clasificación diferente) se pueden ver en la siguiente captura de pantalla, correspondiente a la Figura 29:

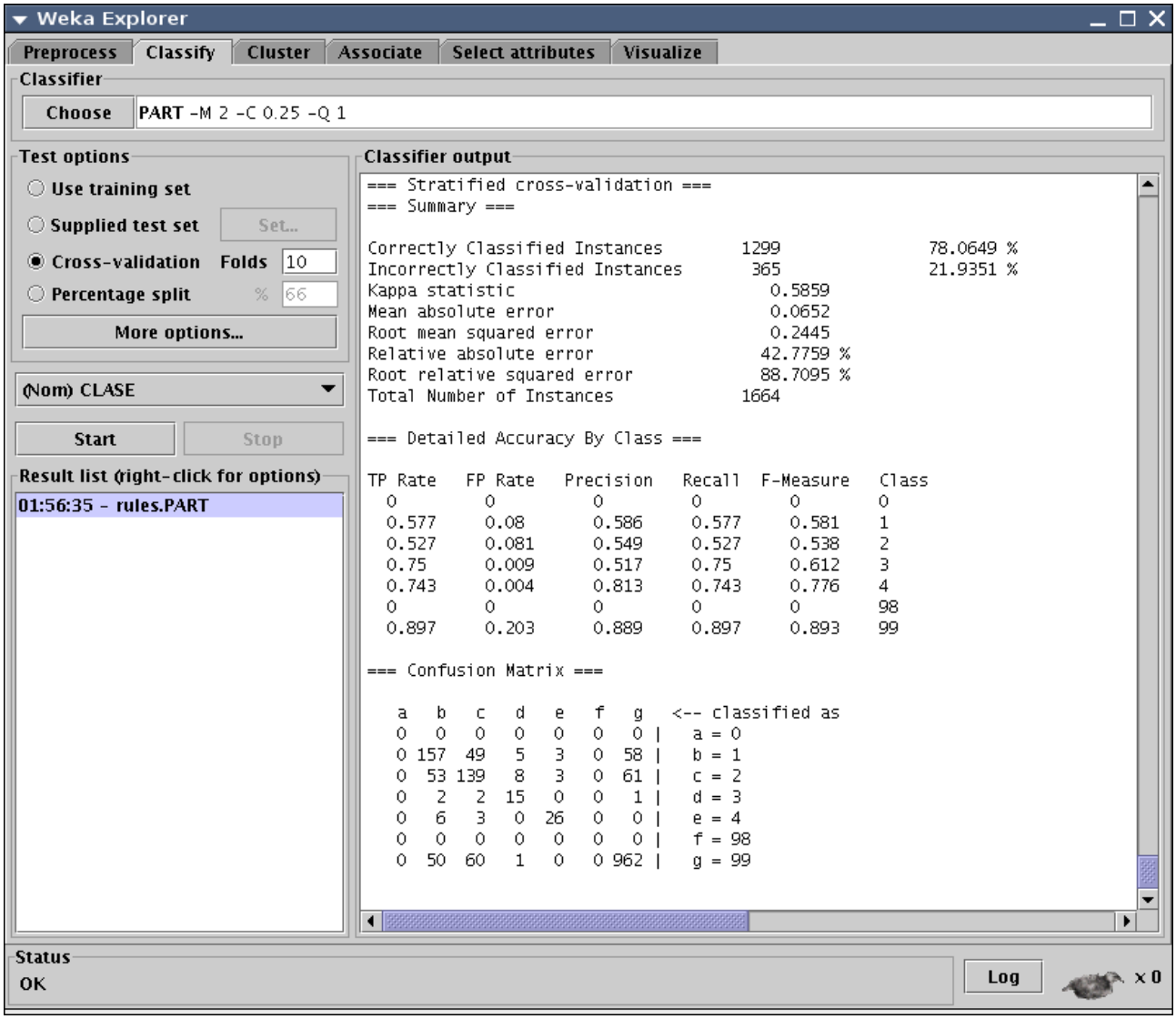


Figura 29: Weka Explorer: resultados del clasificador PART

Al ejecutarlo genera un conjunto de reglas, que más tarde serán incorporadas en el propio cliente. Las reglas tienen una estructura similar a ésta, mostrándose una serie de operadores condicionales sobre los diferentes atributos para obtener la clase a clasificar (en este caso está en formato numérico, la primera regla del ejemplo que se encuentra a continuación es para clases de tipo 1, es decir, *dash*, y la segunda para clases de tipo 99, es decir, *desconocida*):

```
BallSpeed2 > 1.17 AND
BallX2 <= 47.93 AND
BallKickableForOpponent = 0 AND
DESP_My > 0 AND
OpponentPositionValid2 > 0.95 AND
MySpeed <= 0.02: 1 (88.0/1.0)

BallDistance <= 3.48 AND
BallMoving2 = 1 AND
Dif_MyBA <= 12.5 AND
BallX > 33.27 AND
Dif_OS <= 0.03: 99 (219.0/2.0)
```

El conjunto de reglas es variable, el ejemplo anterior constaba de un total de 84 reglas. Además de dichas reglas se obtiene un resumen con el porcentaje de aciertos, la precisión obtenida y la matriz de confusión:

```
Correctly Classified Instances        1299               78.0649 % (instancias
correctamente clasificadas)
Incorrectly Classified Instances       365               21.9351 % (instancias
incorrectamente clasificadas)
Kappa statistic                          0.5859 (medida del coeficiente Kappa)
Mean absolute error                      0.0652 (coeficiente de correlación)
Root mean squared error                  0.2445 (RMSE: Raíz del error cuadrático de
la media)
Relative absolute error                 42.7759 % (MAE / error obtenido procedente
de aplicar el clasificador ZeroR en los datos)
Root relative squared error             88.7095 % (RMSE / error obtenido procedente
de aplicar el clasificador ZeroR en los datos)

Total Number of Instances             1664

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0         0          0         0         0        0 (none)
  0.577     0.08       0.586     0.577     0.581    1 (dash)
  0.527     0.081      0.549     0.527     0.538    2 (turn)
  0.75      0.009      0.517     0.75      0.612    3 (kick)
  0.743     0.004      0.813     0.743     0.776    4 (catch)
  0         0          0         0         0        98 (no_hay)
  0.897     0.203      0.889     0.897     0.893    99 (descono)


=== Confusion Matrix ===

   a   b   c   d   e   f   g   <-- classified as
   0   0   0   0   0   0   0 |   a = 0 (none)
   0 157  49   5   3   0  58 |   b = 1 (dash)
   0  53 139   8   3   0  61 |   c = 2 (turn)
   0   2   2  15   0   0   1 |   d = 3 (kick)
   0   6   3   0  26   0   0 |   e = 4 (catch)
   0   0   0   0   0   0   0 |   f = 98 (no_hay)
   0  50  60   1   0   0 962 |   g = 99 (descono)
```

El porcentaje de aciertos obtenido fluctúa entre 60 y 70% con los parámetros por defecto, aunque se puede aumentar hasta el 80-90% eliminando parámetros o utilizando otros clasificadores.

Para agilizar el procesamiento, hay una serie de filtros que pueden aplicarse, como son:

- *weka.filters.unsupervised.instance.Randomize-S42*
    - Ordena de forma aleatoria las instancias
- *weka.filters.unsupervised.instance.RemovePercentage-P80*
    - Elimina un porcentaje aleatorio de instancias
- *weka.filters.unsupervised.instance.RemoveWithValues-S0.0-Clast-L6*
    - Elimina instancias con valor 0 en algún atributo concreto

Se han realizado diferentes pruebas con filtros buscando lograr una ganancia en el porcentaje de acierto en la clasificación, sin éxito. Ahora bien, la aplicación de los filtros superiores logra obtener mejorías en el tiempo de procesamiento requerido para crear los clasificadores, puesto que debido al alto número de instancias el proceso puede alargarse de unos pocos minutos a media hora en algunos casos concretos.

B.4.5.2 FICHERO .ARFF DE TIPO DASH/TURN/KICK POWER/KICK ANGLE

Aparte del tipo de fichero principal.arff existe otro tipo de ficheros *.arff*, los ficheros generados que contienen información sobre la clase *dash*, sobre la clase *turn*, sobre la clase *kick power*, o sobre la clase *kick angle*, según los archivos resultantes de utilizar el programa *makeArff*. Estos tipos de ficheros se utilizan para predecir los parámetros de la acción que lleva a cabo el portero en el caso de que ésta sea avanzar o girar.

Estos archivos tienen un formato diferente ya que la clase no es un atributo nominal sino que tiene un formato numérico, esto es debido a que el parámetro que se desea clasificar no es el tipo de acción (parámetro que es conocido de antemano), sino el valor de esa acción. Esta afirmación es comprensible si se piensa que lo que se clasifica es la fuerza de avance del portero en caso de utilizar la clase *dash*, y el ángulo de giro de éste para la clase *turn*. En el caso de las clases de tipo *kick*, el valor es la fuerza en el caso de *kick power*, y el ángulo en el caso de *kick angle*. Al utilizar una clase de formato numérico deja de poder utilizarse el clasificador PART, con lo que o bien se puede aplicar un filtro para discretizar los parámetros, o bien se puede utilizar otro algoritmo de ordenación.

Al abrir el fichero, *Weka* nos mostrará la imagen de la Figura 30 al pulsar sobre el atributo CLASE:

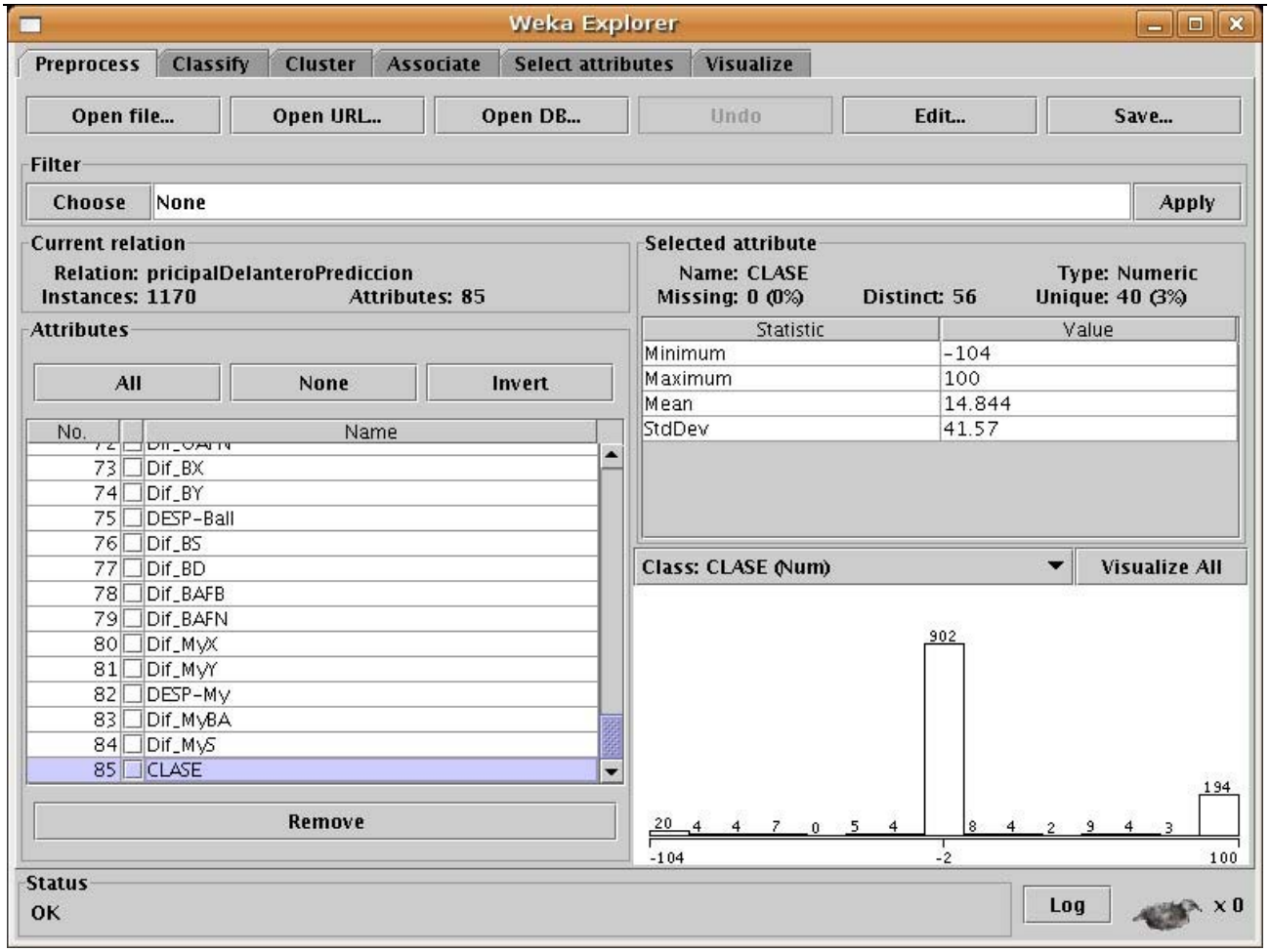


Figura 30: Weka explorer: clase del fichero de tipo dash/turn

Se observa que la representación gráfica de la clase por instancia difiere ampliamente de la que mostraba el fichero principal, puesto que al ser numérica se muestra únicamente en un rango de valores en los que se ajustan las diferentes instancias.

El algoritmo de clasificación que se utiliza es el **clasificador M5**. Este algoritmo funciona de manera diferente al PART, ya que el tipo de árbol que genera es un árbol de modelos de regresión. Esto quiere decir que en las hojas del árbol hay un modelo de regresión lineal para predecir el valor de la clase de las diferentes instancias que llegan a esa hoja. Los árboles generados son relativamente similares a los obtenidos con el clasificador PART, escogiendo en cada nodo el atributo que minimiza la variación en los valores de la clase que resulta por cada rama. Para su utilización se lanza el clasificador con las opciones por defecto en el *Weka*, como se muestra en la Figura 31.

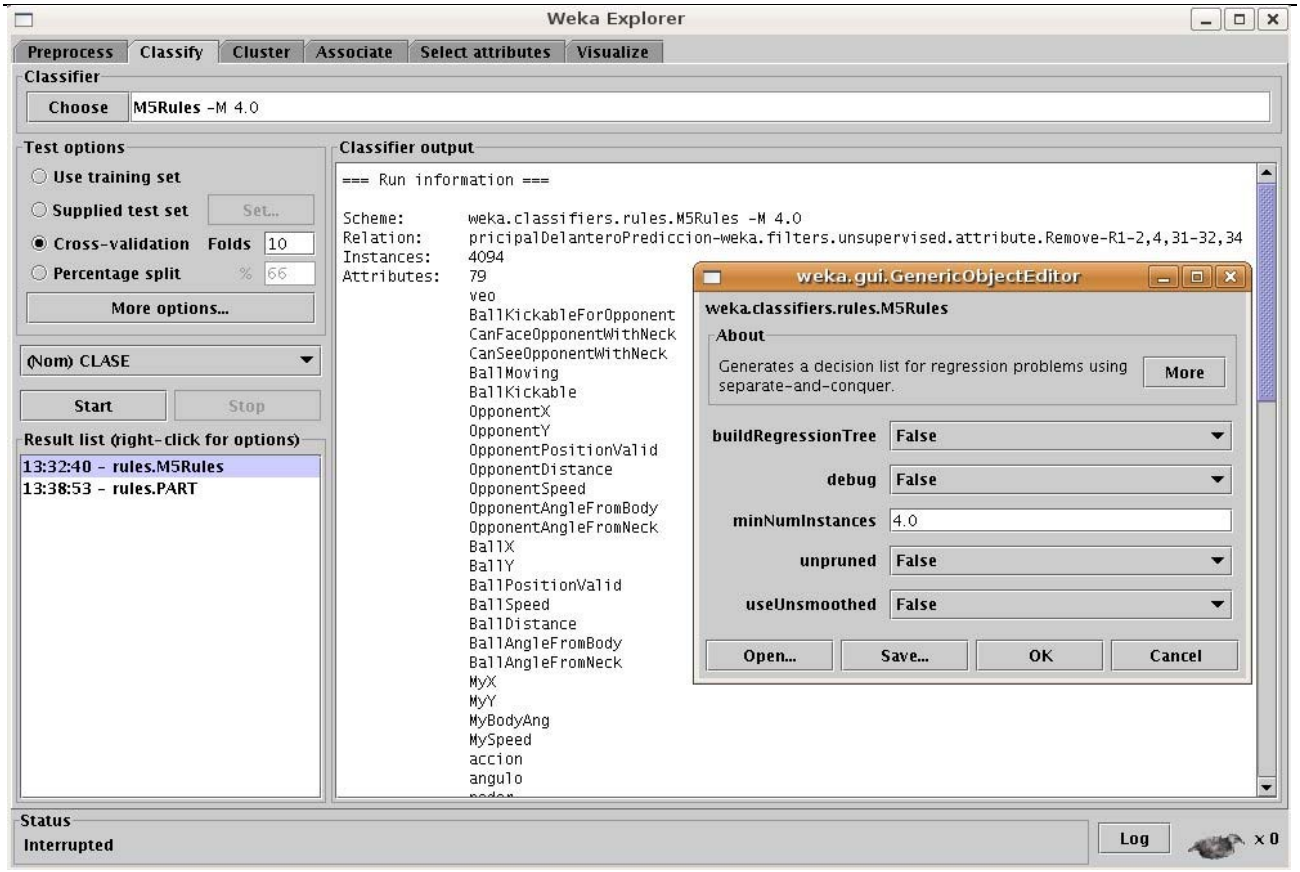


Figura 31: Weka explorer: clasificador M5

En la imagen superior se puede observar que, en el nombre de la relación, previamente se ha utilizado el filtro no supervisado de eliminación de atributos, para eliminar los atributos que no son útiles y que ya han sido previamente mencionados.

Este clasificador genera un árbol, que se puede ver a continuación:

```
M5 pruned model tree:
(using smoothed linear models)

BallSpeed2 <= 0.505 :
|   OpponentDistance <= 34.28 :
|   |   BallDistance2 <= 1.565 :
|   |   |   Dif_BY <= 0.085 : LM1 (134/0%)
|   |   |   Dif_BY >  0.085 :
|   |   |   |   Dif_BAFN <= 1.62 : LM2 (53/0%)
|   |   |   |   Dif_BAFN >  1.62 :
|   |   |   |   |   BallAngleFromBody <= 10.685 :
```

Aparte también genera una serie de reglas (el número de reglas es variable, puesto que el conjunto de instancias con el que se trabaja es enorme, hay multitud de atributos, y no hay que olvidar el alto componente aleatorio de todo este proceso), cada regla da el valor de los parámetros de la clase correspondiente mediante operaciones aritméticas entre sus atributos:

```
LM num: 1
CLASE =
   -0.0002 * server
   + 0.4119 * veo=0
```

```
   - 0.1424 * OpponentX
   - 0.2508 * OpponentY
   - 2.8955 * OpponentPositionValid
   - 0.0652 * OpponentDistance
   + 16.605 * OpponentSpeed
   + 0.0295 * OpponentAngleFromBody
   - 0.0042 * OpponentAngleFromNeck
   + 0.1272 * BallX
   + 0.1205 * BallY
   + 0.4377 * BallSpeed
   + 0.0107 * BallDistance
   - 0.0135 * BallAngleFromBody
...
```

Y por último devuelve el resumen con el coeficiente de correlación dado.

```
=== Summary ===

Correlation coefficient                  0.5322 (coeficiente de correlación)
Mean absolute error                     18.4189 (MAE: error absoluto de la media)
Root mean squared error                 36.0171 (RMSE: Raíz del error cuadrático de
la media)
Relative absolute error                 62.1871 % (MAE / error obtenido procedente
de aplicar el clasificador ZeroR en los datos)
Root relative squared error             86.5599 % (RMSE / error obtenido procedente
de aplicar el clasificador ZeroR en los datos)
Total Number of Instances             1170
```

El coeficiente de correlación muestra la capacidad de clasificación de estas reglas generadas en comparación con las instancias reales, tomando valores entre -1 y 1, siendo -1 una correlación lineal inversa, con 0 significando que las relaciones no tienen ningún punto en común, y tomando 1 como una correlación positiva perfecta. Así pues interesa que los valores de los índices de correlación sean positivos y cuanto más cerca posible del 1 mejor, puesto que implicaría que la función que representa las reglas obtenidas con el M5 clasifican a la perfección la fuerza de avance o el ángulo de giro del rival.

B.4.6 INCLUSIÓN DE LOS FICHEROS DE REGLAS EN EL ETIQUETADOR

El siguiente paso es utilizar los programas *Part2C* y *Rule2C* para incorporar las reglas obtenidas por los clasificadores al fichero *Reglas.C*. Estos programas toman como entradas las reglas obtenidas en los anteriores clasificadores y las convierten a formato C conformando el etiquetador de formar que sea legible por el delantero.

El programa *Part2C* toma como entrada los resultados obtenidos por el clasificador aplicado al fichero de datos de tipo principal, así pues toma como entrada las reglas generadas por el clasificador PART, de ahí su nombre.

En cambio el programa *Rule2C* utiliza los resultados del clasificador aplicado a los ficheros de datos de tipo *dash*, de tipo *turn*, de tipo *kick power*, o de tipo *kick angle*, tomando este programa como entrada las reglas generadas por el clasificador M5.

Para ejecutar estos programas se utilizan las siguientes llamadas, siempre desde consola:

- *part2C -i fichero_reglas_weka -o fichero_salida*
    - *part2C -i vw_part_principal.txt -o vw_etiquetador_principal.txt*
- *rule2C -i fichero_reglas_weka -o fichero_salida*
    - *rule2C -i vw_m5rules_dash.txt -o vw_etiquetador_dash.txt*

A continuación se muestra un ejemplo de fichero de reglas emitido por el algoritmo M5 al clasificar un archivo de tipo *dash*. Las reglas generadas en un fichero de tipo *turn* son exactamente iguales, puesto que provienen del mismo clasificador, al igual que las reglas provenientes de los ficheros de tipo *kick power* o *kick angle*:

```
=== Run information ===

Scheme:       weka.classifiers.rules.M5Rules -M 4.0
Relation:     principalDelanteroPrediccion-
weka.filters.unsupervised.attribute.Remove-R1-2,4,31-32,34
Instances:    708
Attributes:   79
              veo
              BallKickableForOpponent
              CanFaceOpponentWithNeck
...
              Dif_MyBA
              Dif_MyS
              CLASE
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

M5 pruned model rules
(using smoothed linear models) :
Number of Rules : 18

Rule: 1
...
Rule: 16
IF
   OpponentX <= 45.715
   BallY <= 5.71
THEN

CLASE =
   2.8932 * OpponentX
   - 2.9851 * OpponentY
   - 1.3856 * BallY
   - 103.6024 [13/75.26%]

Rule: 17
IF
   OpponentX > 45.715
THEN

CLASE =
   22.409 * OpponentX
   - 974.0965 [9/0.482%]
```

```
Rule: 18

CLASE =
   - 100 [6]



Time taken to build model: 10.93 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.4759
Mean absolute error                     31.2275
Root mean squared error                 56.1618
Relative absolute error                 71.6824 %
Root relative squared error             90.3704 %
Total Number of Instances              708
```

Se puede observar que el clasificador habrá creado una serie de reglas con una sentencia *IF* y una sentencia *THEN*, que viene a decir que si se cumple la condición del IF, entonces la clase será lo que devuelva la sentencia *THEN*. En el ejemplo se ha puesto un fichero de reglas de la clase *dash*, así que el resultado es numérico.

A continuación se muestra una regla obtenida a través del clasificador PART proveniente de un fichero de tipo principal:

```
BallX <= 41.99 AND
BallDistance2 <= 3.56 AND
CanFaceOpponentWithNeck = 1 AND
BallPositionValid2 <= 0.86 AND
MyY2 > -13.51: 2 (turn)
```

Se observa que no utiliza las sentencias *IF* condición *THEN* resultado, sino que se muestra una lista de condiciones que dan lugar a una clase concreta (en el caso del ejemplo la clase es 2, es decir, *turn*)

Una vez estudiados los resultados obtenidos a través de los diferentes clasificadores de *Weka*, el siguiente paso es observar un ejemplo del archivo de reglas devuelto por el *Rule2C* o el *Part2C*, que ofrecen resultados completamente idénticos:

```
if ( (OpponentDistance2 > 36.62) && (OpponentX > 47.725) ){
...
if ( (OpponentX <= 45.715) && (BallY <= 5.71) ){
return (2.8932 * OpponentX) -  (2.9851 * OpponentY) -  (1.3856 * BallY) - 103.6024
);
}
if ( (OpponentX > 45.715) ){
return (22.409 * OpponentX) - 974.0965 );
}
return (- 100);
```

Se puede ver que son las reglas anteriormente expuestas en el caso del clasificador del archivo con datos de *dash* en un formato comprensible para el lenguaje C, por lo que podrán incorporarse a los ficheros de los que consta el cliente.

Debido a ciertas incompatibilidades en las reglas generadas por los clasificadores de *Weka* a veces surgen reglas mal formadas, proceso que se puede comprobar a la hora de compilar el conjunto. Concretamente se ha experimentado que en algunas ocasiones la operación lógica OR entre varias posibilidades para un mismo atributo ha dado problemas, representándose las diferentes opciones separadas por comas en vez de con el operador lógico "||". Este problema se resuelve utilizando el programa *ParseReglas*, que recorre los ficheros de reglas generados por los programas *Rule2C* y *Part2C* solventando este problema.

Un ejemplo de mala localización es el siguiente:

```
if ( (MyAccion >= 2,3,99) ){
return ((DESP_O * 1.475) – (4.756 * BallX)
```

Al aplicar el programa parseReglas quedaría de la siguiente manera:

```
if ( ((MyAccion >= 2) || (MyAccion >= 3) || (MyAccion >= 99)) ){
return ((DESP_O * 1.475) – (4.756 * BallX)
```

La manera de invocar el programa *parseReglas* es de la siguiente forma:

- *parseReglas -i fichero_salida_part2C/rule2C -o fichero_salida*
    - *parseReglas -i vw_etiquetador_principal.txt -o vw_etiquetador_principal_reglas.txt*

El último paso, una vez creado el etiquetador, es incorporar estas reglas al fichero *Reglas.C*, proceso que se hace a mano y que es tan simple como copiar y pegar cada uno de los tres ficheros de reglas obtenidos en su correspondiente función:

- función *PositionInfo::dameClase* recibe las reglas generadas en el etiquetador principal, obtenidas a través del clasificador PART y que han sido tratadas por el programa *Part2C*.
- función *PositionInfo::dame_dash_power* recibe las reglas generadas en el etiquetador del fichero de *dash*, de la fuerza de avance del rival, obtenidas a través del clasificador M5 y que han sido tratadas por el programa *Rule2C*.
- función *PositionInfo::dame_turn_angle* recibe las reglas generadas en el etiquetador del fichero de *turn*, del ángulo de giro del rival, obtenidas a través del clasificador M5 y que han sido tratadas por el programa *Rule2C*.
- función *PositionInfo::dame_kick_power* recibe las reglas generadas en el etiquetador del fichero de *kick power*, de la fuerza del golpeo del rival, obtenidas a través del clasificador M5 y que han sido tratadas por el programa *Rule2C*.

- función *PositionInfo::dame_kick_angle* recibe las reglas generadas en el etiquetador del fichero de *kick angle*, del ángulo de golpeo del rival, obtenidas a través del clasificador M5 y que han sido tratadas por el programa *Rule2C*.

No hay que olvidarse de recompilar el agente una vez terminado este proceso, basta con ejecutar desde la consola el comando *make* en el directorio raíz del agente CMUnited 99.

B.5 CREACIÓN DEL MODELADOR

Como se explicó en el punto 3.2.3, el proceso de creación del modelador consta de varios puntos, entre ellos la obtención de datos inicial prescindiendo de los datos proporcionados por el *Soccer Server* y posterior tratamiento con la intención de obtener unas reglas que determinen las acciones a llevar a cabo por el agente rival para posteriormente incorporarlas al propio agente. La Figura 32 muestra el esquema global de implementación del modelador a través de un diagrama de flujo de datos con nombres de ficheros y programas reales:

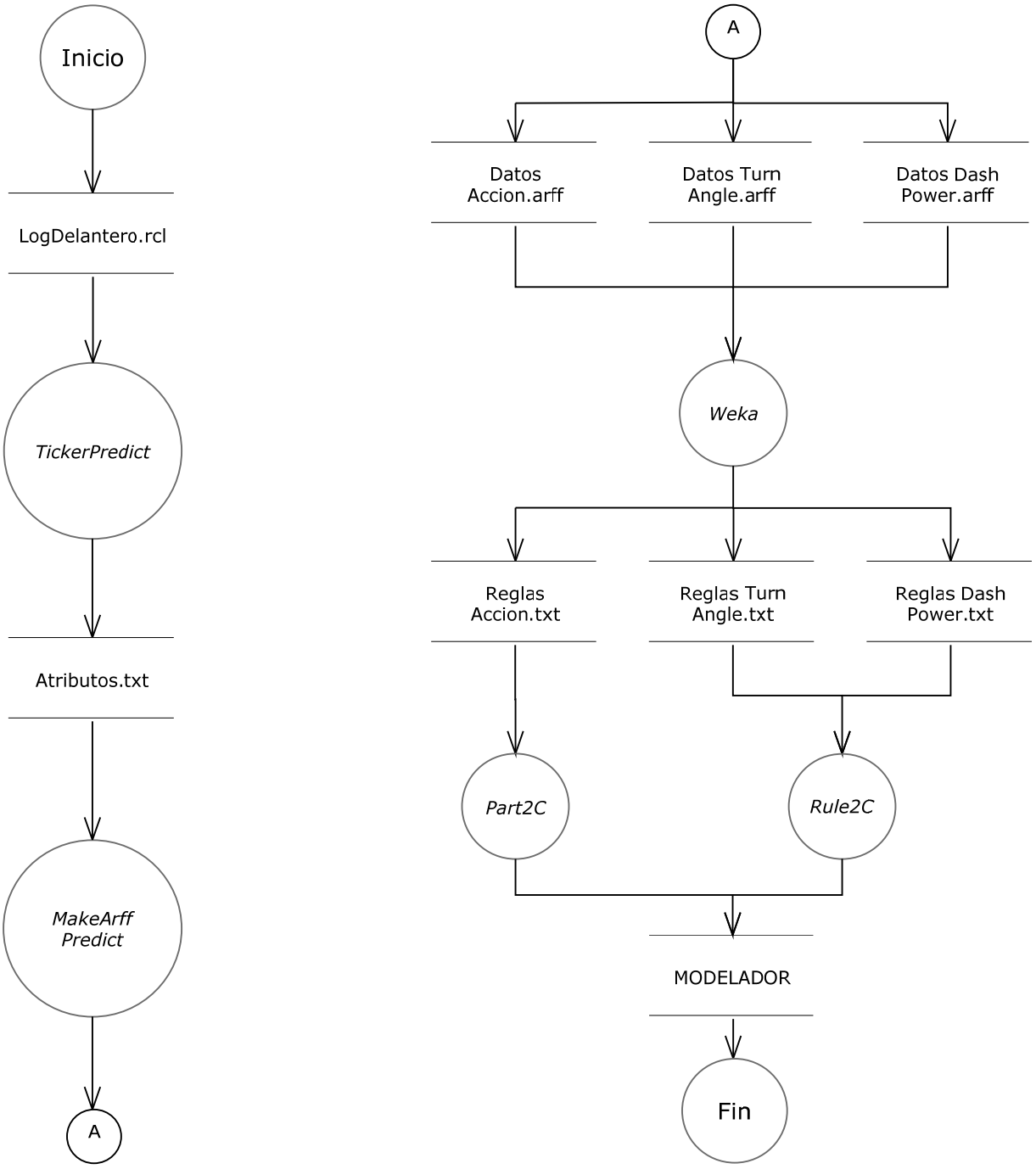


Figura 32: Esquema de implementación del modelador

La condición indispensable para generar el modelador es que antes se haya generado el etiquetador y se haya incorporado al agente CMUnited 99. En caso contrario los datos obtenidos no tendrán ningún tipo de validez y no se podrán aplicar las utilidades listadas a continuación.

Para utilizar el modelador hay que asegurarse de establecer a un valor determinado varios parámetros concretos del archivo de configuración *config_params.conf* (se puede consultar el manual de referencia para ver las distintas posibilidades y comprender la utilidad de cada parámetro). Los atributos son los siguientes:

1. *FLAG_ETIQ*: establecerlo a falso, es decir, a 0.
2. *KICK_MODEL*: con valor 0, es decir, sin utilizar ningún modelo.
3. *NUM_MOVEMENTS*: con valor 50, para que al igual que en el proceso de creación etiquetador se realice una simulación con 50 ejecuciones.

Como se ha expuesto anteriormente, el proceso de generación del etiquetador y el del modelador tienen en común ciertos puntos. En los siguientes puntos se explicará en detalle los elementos diferenciadores y los procesos a tener en cuenta. Los puntos de los que se compone esta operación son los siguientes:

- Obtención de datos.
- Generación de un fichero con los datos percibidos por el agente CMUnited 99.
- Generación de los archivos propios de *Weka* y posterior procesamiento.
- Inclusión de los ficheros de reglas en el etiquetador.

B.5.1 OBTENCIÓN DE DATOS

El proceso de generación del modelador precisa de una toma previa de datos en forma de simulación de partidos, al igual que a la hora de generar el etiquetador. Sin embargo, la inclusión del etiquetador a través del fichero Reglas.C provocará que los resultados obtenidos sean diferentes.

La simulación ha de constar de dos simulaciones completa de 50 jugadas (parámetro *NUM_MOVEMENTS* en config_params.conf) que proporcionan 100 jugadas de ataque a estudiar. Este tamaño de muestra es análogo al utilizado en la generación del etiquetador.

Al simular partidos también se generan archivos de logs, pero debido al parámetro *FLAG_ETIQ* igual a falso los datos registrados son diferentes. El formato de los datos generados por el fichero *log-oppnt.txt* es algo más corto que el generado en el etiquetador. Dicho formato es el siguiente: [9 enteros] [18 dobles] acción [2 dobles], conservando los datos percibidos por el delantero y la acción que ha predecido el clasificador ya incorporado (junto con sus parámetros).

Además existe la peculiaridad de que a la hora de generar el modelo no se puede contar con los datos precisos de las acciones del portero obtenidos a través de los logs generados por el servidor, puesto que se va a contar con el etiquetador generado anteriormente para decidir cuál va a ser la clase del agente rival.

B.5.2 GENERACIÓN DE UN FICHERO CON LOS DATOS SENSADOS POR EL AGENTE CMUNITED 99

A través de la herramienta *Ticker_predict*, se combinan entre sí los datos del fichero de *log* generado por el agente delantero con los datos observados por sí mismo, obteniendo trazas de información que contienen los datos registrados en el instante *t* y en el instante *t-1* junto con la acción a seguir.

El programa *Ticker_predict* tiene un comportamiento algo diferente al ticker, pero en esencia realizan la misma función. La llamada es algo diferente porque se le pasa como parámetro únicamente el fichero de *log* creado por el cliente CMUnited 99:

- *ticker_predict -i log_oponente_CMU -v tamaño_ventana -o salida*
    - *./ticker_predict -i CMUnited1-l-oppnt_modelador.log -v 2 -o salida_ticker_predict.txt*

El fichero de salida del *Ticker_predict* muestra similitudes con el fichero que se obtiene a través del programa *Ticker*, pero tienen una configuración de parámetros diferente. A continuación se muestra unos cuantos grupos de datos de la traza obtenida:

```
49, 0, 0, 1, 0, 1, 1, 1, 0, 49.05, -0.03, 0.98, 13.60, 0.02, -3.35, -21.35, 38.23,
1.29, 0.90, 2.54, 2.95, 17.67, -0.33, 35.45, 0.29, 2.00, 0.16, 3, 23.77, 100.00, 99
(clase desconocida), 0.00, 0.00

50, 0, 1, 1, 0, 1, 1, 1, 0, 48.99, 0.05, 0.99, 13.23, 0.04, -2.77, -20.77, 40.79,
2.44, 0.95, 2.45, 5.46, 21.43, 3.43, 35.76, 0.30, 1.67, 0.06, 2, -0.59, 0.00, 1
(clase dash), 0.00, 99.72 (fuerza de avance)

51, 0, 1, 1, 0, 1, 1, 1, 0, 49.25, 1.00, 0.99, 13.50, 0.10, -2.43, -20.43, 42.98,
3.55, 0.95, 2.33, 7.92, 18.85, 0.85, 35.76, 0.30, 5.43, 0.03, 2, 4.46, 0.00, 1
(clase dash), 0.00, 100.01 (fuerza de avance)

55, 0, 0, 1, 0, 1, 1, 1, 0, 48.74, 3.84, 0.98, 13.51, 0.06, 9.07, -8.93, 50.27,
6.87, 0.90, 1.73, 15.99, 18.12, 0.12, 35.70, 0.29, 6.17, 0.00, 2, -0.24, 0.00, 2
(clase turn), 17.03 (ángulo de giro), 0.00
```

Como se ve es más sencilla que la traza obtenida en el etiquetador, puesto que no se necesitan tantos datos. Concretamente el formato de la traza es *[9 enteros] [18 dobles] acción [2 dobles]*, siendo los primeros atributos los datos percibidos por el agente, y siendo la acción y sus parámetros obtenidos a través del etiquetador.

B.5.3 GENERACIÓN DE LOS ARCHIVOS PROPIOS DE WEKA Y POSTERIOR PROCESAMIENTO

Este apartado es exactamente análogo al apartado del mismo nombre del etiquetador, excepto por la salvedad del programa que recoge la salida del *Ticker_predict*, en este caso se llama *MakeArff_predict*, y toma un fichero de encabezado diferente para adaptarse a la nueva información de que se dispone, puesto que los atributos utilizados en el etiquetador y en el modelador son diferentes, siendo la principal diferencia que en el etiquetador se usan datos observados por el delantero junto con datos verídicos obtenidos del servidor, mientras que en el modelador los datos se basan únicamente en la observación que ha podido llevar a cabo el delantero (aunque hay que recordar que a la hora de modelar el clasificador está incorporado al delantero).

La llamada al *makeArff_predict* es la siguiente:

- *makeArff_predict -i fichero_generado_ticker -e encabezado_predict*
    - *./makeArff_predict -i salida_ticker_predict.log -e encabezado_predict.txt*

En este punto se trata el fichero obtenido con el programa *MakeArff_predict*, que genera archivos comprensibles para la herramienta de minería de datos *Weka*, a través de la cual se obtienen reglas para clasificar tanto la acción a realizar por el agente rival como el ángulo de giro o la fuerza de avance del mismo, además de la fuerza y el ángulo de golpeo, si se diera el caso. Los clasificadores utilizados son los mismos que en el etiquetador, siendo el clasificador PART el elegido para clasificar la acción, y el clasificador M5 cuando se trata del archivo generado con los datos del *turn*, del *dash*, del *kick power* o del *kick angle*.

Conviene indicar que, se han de eliminar los mismos atributos que ya se eliminaron en el etiquetador debido a que no aportan nada a la clasificación, o la entorpecen como es el caso del atributo tiempo. El modo de realizar la eliminación es mediante el filtro no supervisado de atributos *Remove*. Así pues se ha de ejecutar el siguiente filtro: *weka.filters.unsupervised.attribute.Remove-R1-2,4,34-35,37*, que es el mismo que el utilizado en el etiquetador, afectando a los mismos parámetros, pero encontrándose éstos en otras posiciones.

B.5.4 INCLUSIÓN DE LOS FICHEROS DE REGLAS EN EL ETIQUETADOR

Al igual que en el etiquetador, se aplica el programa *Part2C* sobre el fichero de reglas que clasifican la acción del rival, y el programa *Rule2C* sobre los ficheros de reglas que clasifican el ángulo de giro o la fuerza del avance del portero rival cuando se determina que esa es la clase que realiza. No hay ninguna diferencia puesto que los ficheros de reglas conservan el mismo formato independientemente de los atributos de los que consten los ficheros de entrada, puesto que se ha utilizado el mismo clasificador tanto en el etiquetador como en el modelador.

Así mismo también es necesario pasar los resultados de ambos ficheros por el programa *ParseReglas*, por si se ha generado alguna incongruencia en el tratamiento de las reglas por parte de *Weka*.

Estos ficheros de reglas se incluyen a mano en el fichero *Prediccion.C*, que tiene un formato análogo al *Reglas.C* anteriormente visto. Al hacerlo hay que recordar que este fichero no es un archivo de configuración externo, y por tanto hay que compilar de nuevo el agente CMUnited 99. La compilación se hace del mismo modo que en el proceso de creación del etiquetador, con el comando *make* ejecutado desde el directorio donde se encuentra el cliente CMUnited 99.

Anexo C MANUAL DE REFERENCIA

Desde el principio del proyecto se tuvo en cuenta que todo lo que se ha hecho podría ser utilizado en un futuro para continuar con la investigación en este tema. Como se pueden leer en las líneas de trabajo futuras, es muy sencillo continuar con el proyecto, únicamente habría que añadir más agentes rivales para avanzar un paso más en el aprendizaje del agente delantero. Por tanto, es importante que todos los programas tengan una opción de escalabilidad, y que estén bien comentados para poder ser continuados por otras personas ajenas a este proyecto.

El agente delantero se basa casi por completo en el agente que presentó la universidad Carnegie Mellon para la competición del año 1999. En Ledezma [3] se introdujeron varios cambios para aportar un sistema de aprendizaje, cambios que han sido respetados en la mayoría y actualizados en alguna que otra circunstancia. La mayor parte de la funcionalidad está previamente especificada en puntos anteriores de este proyecto, pero hay varios elementos que conviene que sean referenciados en profundidad.

C.1 LISTADO DE FICHEROS

A continuación se muestran los diferentes ficheros de los que cuenta tanto el delantero (cliente CMUnited 99) como el *Trainer*, y una explicación sencilla de la funcionalidad de cada uno de ellos.

C.1.1 FICHEROS DEL CLIENTE CMUNITED 99

Este punto enumera los ficheros de los que se compone el cliente CMUnited 99.

- *arbol.C*: reglas generadas para el modelo basado en árbol de decisión, no utilizado en este proyecto.
- *behave.C*: ejecutado por cada ciclo del servidor, determina la acción enviada al servidor.
- *client.C*: el ciclo principal de ejecución, encargado de actualizar los estados de los objetos, parsea los elementos percibidos, guarda la información percibida en variables temporales, y elige las comunicaciones a realizar.
- *dribble.C*: contiene información para regatear en posesión de la bola.
- *geometry.C*: define funciones para realizar razonamientos basados en la geometría en el campo de juego.
- *kick.C*: contiene las funciones necesarias para chutar la bola.
- *MemAction.C*: contiene funcionalidades para interceptar la bola.
- *MemOption.C*: almacena los parámetros de inicialización, y también los parámetros leídos del cliente y del servidor.
- *Memory.C*: contiene la definición de la clase Memoria a alto nivel.
- *MemPlayer.C*: almacena la información referente al cliente, su posición en el campo, velocidad, energía, etc.

- *MemPosition.C*: almacena la información referente a los otros objetos del campo, como otros jugadores, la bola, los marcadores del campo, etc.
- *netif.C*: maneja la comunicación con el servidor, siempre a través de *sockets UDP*.
- *parametros.C*: reglas para el modelo árbol, no utilizado en este proyecto.
- *parse.C*: parsea los mensajes entrantes del servidor.
- *prediccion.C*: donde se almacenan las reglas generadas en el proceso de generación del modelador.
- *reglas.C*: donde se almacenan las reglas generadas en el proceso de generación del etiquetador.
- *test.C*: contiene funciones para probar las distintas habilidades del cliente.
- *utils.C*: implementación de utilidades genéricas utilizadas en cualquiera de los archivos anteriores.

Además hay que contar con los siguientes ficheros:

- Ficheros *Makefile*: si se desean añadir más archivos habría que editar las opciones de compilación en este fichero.
- *charliclient*: ejecutable generado, es el que se utiliza para lanzar el programa.
- Directorio *Logfiles*, que contiene los ficheros de *log* generados por el delantero

C.1.2 FICHEROS DEL TRAINER

Este punto enumera los ficheros de los que se compone el *Trainer*.

- *client.C*: el ciclo principal de ejecución del *Trainer*, inicializa la memoria, parsea los mensajes del servidor, coloca a los jugadores y a la bola, y se comunica con el servidor.
- *data.C*: clases relacionadas con el almacenamiento de datos estadísticos de partidos.
- *epoch.C*: implementación del controlador de épocas, que controla los ciclos que han pasado, almacena la información generada en cada ciclo, y pasa información a los clientes en cada nueva época.
- *geometry.C*: muy similar al *geometry.C* del cliente, contiene la implementación de funciones comunes relacionadas con estructuras geométricas.
- *MemOption.C*: misma funcionalidad que en el cliente.
- *Memory.C*: misma funcionalidad que en el cliente.
- *MemPosition.C*: misma funcionalidad que en el cliente.
- *MemTrain.*C: contiene la implementación de la clase que administra qué modulo de entrenamiento está activo.
- *netif.C*: misma funcionalidad que en el cliente.
- *parse.C*: misma funcionalidad que en el cliente.

- *TMataque.C*: módulo de entrenamiento para el ataque, es una prueba y no se utiliza.
- *TMbreakaway.C*: módulo de entrenamiento para reiniciar la jugada y colocar a los jugadores en sus posiciones originales, almacena los datos de jugadas, goles, tiros fuera, etc. En su lugar se utiliza *TMmodel2.C*.
- *TMdribble.C*: módulo de entrenamiento para regatear con el balón.
- *TMgame.C*: módulo de entrenamiento para simular el comportamiento de un partido. Se asegura de que el balón no se traba y no está quieto por largos períodos de tiempo.
- *TMgoalie.C*: módulo de entrenamiento para el portero.
- *TMhardkick.C*: módulo de entrenamiento para chutar con fuerza la pelota.
- *TMhardkick2.C*: otro módulo de entrenamiento para chutar con fuerza la pelota, implementación diferente a *TMhardkick.C*.
- *TMintercept.C*: módulo de entrenamiento para interceptar la pelota cuando ésta está en movimiento.
- *TMintercept2.C*: otro módulo de entrenamiento para interceptar la pelota cuando ésta está en movimiento, implementación diferente a *TMintercept.C*.
- *TMkeepaway.C*: módulo de entrenamiento para despejar la pelota.
- *TMmodel.C*: implementación de *TMbreakaway.C* no utilizada.
- *TMmodel2.C*: implementación de *TMbreakaway.C* final, coloca a los jugadores sobre el terreno de juego y almacena toda la información referente a las jugadas.
- *TMsetplay.C*: módulo de entrenamiento para la implementación de las reglas de juego, esto es, pitido inicial, saques de puerta, córners, saques de banda, etc.
- *TMshot.C*: módulo de entrenamiento de tiro.
- *TMtest.C*: módulo de entrenamiento de prueba, no utilizado.
- *TMturnball.C*: módulo de entrenamiento para el comportamiento de girar con la pelota controlada.
- *TMturnball2.C*: otro módulo de entrenamiento para el comportamiento de girar con la pelota controlada, distinto a *TMturnball.C*.
- *utils.C*: misma funcionalidad que en el cliente.

Además hay que contar con el fichero binario que se genera al compilar correctamente el *Trainer*:

- *trainer*: ejecutable generado, es el que se utiliza para lanzar el programa.

C.1.3 OTROS FICHEROS

Este punto enumera el resto de ficheros que se encuentran disponibles, aparte del cliente CMUnited 99 y el *Trainer*.

- Ficheros *Makefile*: si se desean añadir más archivos habría que editar las opciones de compilación en este fichero.

- Directorio *TrainingData*, que contiene los ficheros de resultados obtenidos a la hora de simular.
- Directorio *shared*, con archivos comunes al *coach* o entrenador (no utilizado en este proyecto), y que contiene archivos que serán importados a la hora de compilar el *Trainer*. Dichos archivos son los siguientes:
    - *data.C*, *geometry.C*, *MemPosition.C*, *netif.C*, *utils.C*: ya comentados en el *Trainer*, las modificaciones en estos archivos han de realizarse en estos archivos del directorio *shared*.
    - *opt-array.C*, *opt-code.C*, *opt-def.C:* definición de parámetros, de funciones y de valores para el archivo *MemOptions.C* del *Trainer*, a la hora de compilarlo se incluyen estos ficheros.

Además se cuenta con dos ficheros de script para el lanzamiento de todo el proceso de forma automática, dichos ficheros son los siguientes:

- *E100_AICOM2.sh*.
- *ATAQUE_AI2.sh*.

El siguiente punto describe en profundidad estos scripts.

C.2 SCRIPTS UTILIZADOS

Se han utilizado dos scripts de shell a la hora de realizar las simulaciones, el script *E100_AICOM2.sh* y el script *ATAQUE_AI2.sh*. A continuación se puede ver una explicación de dichos scripts y su código fuente.

La forma de lanzar las ejecuciones es a través del fichero de script ***E100_AICOM2.sh***, que se encarga de cargar el servidor, los dos agentes y el *Trainer*, y que almacena los resultados en un fichero.

Para ejecutar el script hay que realizar la siguiente llamada desde el directorio del *Trainer*:

- *./E100_AICOM2.sh –f fichero_de_resultados –i número_iteraciones*
    - *./E100_AICOM2.sh –f trilearn_resultados.txt –i 100*

El número de iteraciones es opcional, si no se pasan como parámetro se toma el valor por defecto de cien. El script es muy simple y tiene el siguiente formato:

```
#!/bin/bash

show_help ()
{
  if [ $# -gt 0 ]; then
    echo "E100_AICOM2: opción inválida $1"
  fi

  echo "Modo de empleo: ./E100_AICOM2.sh (opciones) [iteraciones]
Opciones:
   -f ARCHIVO [i]     Guarda los resultados de las ejecuciones en ARCHIVO
```

```
        El parámetro opcional 'i' es el número de iteraciones (por defecto 100)
   -h   Muestra este mensaje y finaliza"
}

if [ $# -lt 2 ]; then
  if [ "$1" != "-h" ]; then
   show_help $1
   exit 0
  fi
fi

if [ $# -gt 3 ]; then
  show_help "(demasiados argumentos)"
  exit 0
fi

# Directorio de almacenamiento de Logs
cd /home/alberto/Desktop/PFC/CMU/support_CMU/trainer/TrainingData

case $1 in
    -h )
   show_help
   ;;
    -f )
   if [ $# -ne 3 ]; then
     n=100
   else
     ver_num=$3
     ver_num=$[ver_num+0]

     if [ $ver_num != $3 ]; then
       show_help "(el tercer parámetro no es un número)"
        exit 0
     fi
     n=$3
   fi

   # Para determinar si el tercer parámetro (iteraciones) es un número
   # No hay un isnum en shell, así que se le suma un 0. Si es una letra, se
convierte en 0

   echo --- Experimentos del Delantero ---
   echo --- CMUnited vs UvA Trilearn ----
   # echo "Lectura de ficheros"

   rm  $2.txt
   touch $2.txt

   time=`date | cut -c1-19`

   echo "--------------RESULTADOS TRAINER--------------
------------ $time ------------

TOTAL   GOLES   PORTERO FUERA FALLOS  FALLIDOS" > $2.txt

   echo "----Numero de ejecuciones: $n----"
   for i in `seq 1 $n`;
     do
     echo EJECUCION $i
     cd /home/alberto/Desktop/PFC/CMU/support_CMU/trainer
```

```
    ./ATAQUE_AI2.sh
    cd TrainingData
    grep '[1-9]' model2.log  >> $2.txt

  done
  ;;
   *)
  show_help $1
  ;;
esac
```

Es importante reseñar que las líneas comentadas empiezan con el carácter #. Si se mira el código fuente se ve que simplemente ejecuta 100 veces (o las deseadas) otro programa, el script ***ATAQUE_AI2.sh***, y almacena los resultados en un fichero que se le pasa como parámetro. Este otro script es más completo, y se muestra a continuación:

```
#!/bin/csh -f

set num_epochs = 1
set count = 0
while ($count < $num_epochs)
  echo Beginning epoch $count
  (date)
  #iniciando servidor
  echo Starting Server....
  (rcssserver &)
  sleep 4

  #si se desea se puede ejecutar el monitor para observar los avances
  #(rcssmonitor &)

  #iniciando delantero
  echo Starting CMU Client....
  sleep 3
  (cd /home/alberto/Desktop/PFC/CMU;  ./charliclient -file server.conf -file
client.conf > prueba_logAI.txt &)

  #iniciando portero
  echo Starting OTHER Client....
  sleep 3
  (cd /home/alberto/Desktop/PFC/tdd/trilearn_rc2003_bin; ./trilearn_player -n 1  >
clientelogAI.txt &)
  #(cd /home/alberto/Desktop/PFC/tdd/otros/VirtualWerder; ./agent localhost vwerder
-goalie > clientelogAI.txt &)
  #(cd /home/alberto/Desktop/PFC/YowAI2005; ./player -g  > clientelogAI.txt &)
  #(cd /home/alberto/Desktop/PFC/WrightEagle2005; ./WE2005 -file conf/client.conf -
file conf/server.conf -host localhost -team WrightEagle -goalie > clientelogAI.txt
&)
  #(cd /home/alberto/Desktop/PFC/tokyotech_rc2005_2d/formations; ../helios_player -
g > clientelogAI.txt &)
  #(cd /home/alberto/Desktop/PFC/fcportugal_bin; ./FCPortugalPlayer -file
client.conf -file server.conf -goalie > clientelogAI.txt &)

  sleep 3

  #iniciar trainer en primer plano
  echo Starting Trainer....
```

```
  (cd /home/alberto/Desktop/PFC/CMU/support_CMU/trainer; ./trainer -file
server.conf -file trainer.conf -epochs_to_run 1 -epoch_start $count > trainer.log)

  #cerrando servidor
  echo Trainer Exited, Killing Server....
  kill `ps ax | grep rcss | grep -v grep | awk '{print $1}'`

  #esperar a los clientes a que se cierren
  sleep 5
  echo Client should have exited

  #por si acaso no se cierran los clientes forzar el kill
  kill `ps ax | grep charl | grep -v grep | awk '{print $1}'`
  kill `ps ax | grep trile | grep -v grep | awk '{print $1}'`
  kill `ps ax | grep vwerder | grep -v grep | awk '{print $1}'`
  kill `ps ax | grep WE2005 | grep -v grep | awk '{print $1}'`
  kill `ps ax | grep player | grep -v grep | awk '{print $1}'`
  kill `ps ax | grep agent | grep -v grep | awk '{print $1}'`
  kill `ps ax | grep portugal | grep -v grep | awk '{print $1}'`

  sleep 5

  @ count = $count + 1
end
```

Si se observa con atención se puede ver que es una llamada al servidor, la inicialización del cliente CMUnited 99, la inicialización del cliente rival, y la ejecución del agente supervisor *Trainer*. Una vez terminadas las 30 ejecuciones programadas en el *Trainer*, se cierra todo lo abierto anteriormente.

En la sección #iniciando portero se ejecuta el portero rival que corresponda. El script por defecto está preparado para lanzar el agente UvA Trilearn, con los otros cinco porteros comentados. Las llamadas son las siguientes:

- **UvA Trilearn 2003** : *trilearn_player -n 1*
- **YowAI 2004**: *./player –g*
- **Virtual Werder 2004**: *./agent localhost vwerder –goalie*
- **Wright Eagle 2005**: *./WE2005 -file conf/client.conf -file conf/server.conf -host localhost -team WrightEagle –goalie*
- **Tokio Tech 2005**: *./helios_player –g*
- **FC Portugal 2011**: *./FCPortugalPlayer -file client.conf -file server.conf -goalie*

Un detalle muy importante a tener en cuenta es el de comprobar las rutas, puesto que por defecto estos scripts cuentan con rutas absolutas que pertenecen al ordenador donde se han realizado las pruebas.

C.3 MODIFICACIONES EN LOS AGENTES DE LA ROBOCUP

Se han realizado modificaciones de código sobre el agente CMUnited 99 y sobre el *Trainer*. Estas mejoras han sido especificadas en el punto 3.2.4. Este punto únicamente las enumera, para más información es necesario recurrir al código fuente de los ficheros. Es importante mencionar que todos los ficheros han recibido una serie de modificaciones iniciales para adaptarlos a la versión 4.5.2 y posteriores del compilador GCC, puesto que el desarrollo inicial fue realizado con una versión anterior.

El agente *Trainer* ha sido actualizado para incluir varias modificaciones. La primera actualización consiste en evitar el progresivo cansancio asociado a los agentes jugadores, debido a que cada agente tiene un nivel de *stamina* que disminuye con las acciones que realiza. Este comportamiento ha sido descrito en el punto 3.2.4.3 y se ha solucionado enviando el mensaje "*(recover)*" al servidor después de cada recolocación de jugadores por parte del *Trainer*. Evidentemente esta característica interesa a la hora de realizar simulaciones, puesto que el aprendizaje se realizará en condiciones óptimas, pero no en un entorno real. El código ha sido modificado en el archivo *trainer/client.C* en la función *MovePlayer*. La segunda actualización relativa al *Trainer* es la modificación descrita en el punto 3.2.4.4. Esta modificación se encuentra en el fichero *trainer/TMmodel2.C* en la función *NewSightHandler*.

El agente CMUnited 99 ha sido adaptado para incluir mejoras especificadas en el punto 3.2.4.6. Para ello se han realizado modificaciones en el archivo de código fuente *CMU/kick.C*, y son relativas a un avance más rápido tanto con modelo incorporado como sin él. Además el algoritmo de decisión de tiro también ha sido afinado. Por otra parte, el agente CMUnited 99 ha sido actualizado para incluir la modificación especificada en el punto 3.2.4.7. La adaptación a formato numérico del registro de la acción llevada a cabo por el agente portero se muestra a través de los clasificadores de los que dispone *Weka* en el propio cliente, en el archivo *reglas.C* o en *prediccion.C*. Los cambios necesarios han sido realizados en el fichero *CMU/MemPosition.C*, y han afectado a todos los programas intermedios utilizados tanto en la generación del etiquetador como en la del modelador.